Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Telecomunicación

Trabajo Fin de Grado

Sistemas de Navegación en Plataformas Móviles Mediante Odometría Visual

Autor: César Alejandro Torrealba Vázquez

Tutor/es: Pedro Gil Jiménez

2019

Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Telecomunicación

Trabajo Fin de Grado

Sistemas de Navegación en Plataformas Móviles Mediante Odometría Visual

Autor: César Alejandro Torrealba Vázquez
Tutor: D. Pedro Gil Jiménez

TRIBUNAL:

Presidente: D. Sergio Lafuente Arroyo

Vocal 1º: D. Francisco Javier Escribano Aparicio

Vocal 2º: D. Pedro Gil Jiménez

FECHA: 10 de julio de 2019

Sistemas de Navegación en Plataformas Móviles Mediante Odometría Visual

César Alejandro Torrealba Vázquez

10 de julio de 2019

Resumen

En este documento se presenta un modo de detectar y rastrear características en secuencias de imágenes (vídeos), grabadas por una cámara monocular móvil (localizada en una plataforma móvil). El objetivo último de estas detecciones es el de estimar la trayectoria recorrida por la cámara. También se presentan los cálculos matemáticos detrás de esta estimación.

Se desarrolló un programa principal como parte del proyecto, capaz de rastrear puntos a lo largo de las secuencias, registrando sus trayectorias. Este programa realizado no estima la trayectoria de la cámara.

También se mencionan varios métodos, usados por distintos autores, de interés para la odometría.

Palabras clave: Odometría, Odometría visual, Detección de características, Rastreo de características, Egomoción.

Abstract

This document introduces a way to detect and track features in image sequences (videos), recorded by a mobile monocular camera (placed on a mobile platform). The final scope of those detections is to estimate the trajectory of the camera. The mathematical calculations behind cameras' trajectory estimation are also presented.

One main program was created as part of this project, able to track some points throughout the sequences, registering their trajectories. This developed program does not estimate the camera's trajectory.

Several methods, used by different authors, of special interest for the odometry are also mentioned.

Keywords: Odometry, Visual odometry, Feature detection, Feature tracking, Egomotion.

Resumen extendido

Se realizó un trabajo de investigación para encontrar la mejor manera de ayudar a una plataforma móvil a ubicarse en su entorno, mediante el uso de odometría visual. La odometría visual es la técnica en la que un sistema móvil estima su posición respecto a un entorno mediante secuencias de imágenes. Por esto, colocando una cámara en cualquier plataforma móvil (como podría ser un robot o un automóvil), es posible utilizar las técnicas de odometría visual para ayudar a que la plataforma se oriente.

Como parte del trabajo se buscó información sobre como determinar la trayectoria de una cámara a partir de sus vídeos, observándose que esto se realiza estimando la rotación y traslación que tuvo la cámara de un fotograma a otro. Esta estimación a su vez se realiza fijándose en las características (como un objeto, o una esquina) detectadas en un fotograma, y viendo el desplazamiento que realizó esta característica para llegar a un fotograma posterior. Esta estimación es posible por medio de los métodos y fórmulas presentadas en la Sección 1.2.

Estas mismas fórmulas reflejan que hay muchos tipos de algoritmos de los que dependen, y por tanto no hay una solución única al problema de la ubicación de la cámara. En general, todos los algoritmos de odometría visual que calculan la estimación de la trayectoria usan las características detectadas en las secuencias de imágenes registradas por la cámara. Para hacer la detección de características y después su búsqueda de correspondencias también existen una gran cantidad de algoritmos, lo que dificulta encontrar la solución ideal para la estimación de la trayectoria. Esta fue una motivación más para buscar distintos proyectos que realizasen estimaciones de trayectorias y para informarse sobre los métodos que usaron.

Se destaca principalmente la documentación adquirida de la página web del proyecto KITTI Vision Benchmark Suite. Los integrantes de este proyecto dotaron a un automóvil de unas cámaras con las cuales registraron distintas secuencias de vídeo, que después publicaron en su web. Posteriormente, distintas personas desarrollaron varios métodos que a partir de las grabaciones de estas secuencias, predicen el movimiento que siguió el vehículo que las grabó. La mayoría de estas personas publicaron artículos en los que comentan los métodos para realizar las predicciones, y la página web del proyecto referencia a estos artículos.

Tras hacer una amplia lectura de los artículos presentes en la página web del proyecto (y de varios de los artículos referenciados en estos) y con el objetivo de destacar los métodos más eficientes, se decidió dedicar un capítulo de este documento (el Capítulo 2) para comentar gran parte de los métodos, haciendo énfasis en sus detalles más interesantes

y útiles para la odometría visual.

Entre la numerosa información recopilada se destaca la gran capacidad que tiene la odometría visual de ser combinada con otros tipos de odometría, creando predicciones de trayectos recorridos que poseen una gran fidelidad con el trayecto original. Esto se puede observar en la Figura 1, donde el segundo mejor método (en cuanto a menor porcentaje medio de errores) referenciado en el proyecto KITTI hace una estimación fidedigna del camino recorrido por la cámara. La alta precisión de esta predicción, unida a que puede ser realizada en tiempo real, es una gran muestra del potencial de la odometría visual, y una motivación más para continuar desarrollándola.

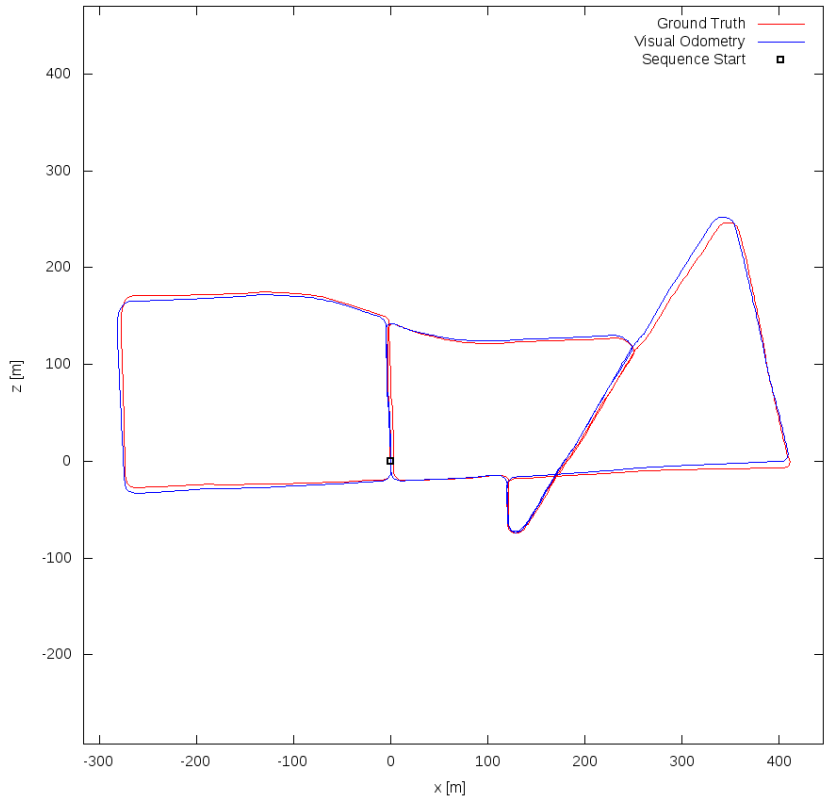


Figura 1: Mapa con la trayectoria recorrida aproximada (de color azul) frente a la realmente recorrida (de color rojo) en la secuencia 13 del proyecto KITTI. La aproximación se realizó con el método V-LOAM. El cuadrado negro marca el inicio del recorrido. Extraido de [1].

Habiendo realizado estas variadas búsquedas de información, se quiso plasmar parte de lo aprendido desarrollando un programa capaz de detectar y rastrear características en secuencias de imágenes, tema al que se le dedica el Capítulo 3 de este documento. El programa creado fue desarrollado con la idea de que luego serviría como base para otro programa, uno capaz de rastrear la trayectoria de una cámara móvil.

Antes de desarrollar el programa se probó a experimentar con varias de las funciones de OpenCV, una librería de funciones de visión por computador. Algunas de estas

funciones eran derivadas de los algoritmos leídos en las publicaciones de la página del proyecto KITTI. Al hacer las pruebas se pudo verificar la eficiencia y limitaciones de varios métodos y algoritmos, y el porqué algunos no son tan o nada utilizados. Algunas de las funciones probadas y no utilizadas en el programa final incluyen una versión del detector de flujo óptico de Lucas-Kanade, del detector de bordes de Canny y del filtro de Kalman.

Como base para el desarrollo del proyecto, se dispuso de un programa que detectaba las características de un fotograma y las rastreaba, buscándoles características correspondientes (el mismo elemento) en los siguientes fotogramas. Este programa registraba todas las posiciones en las que se detectaron las correspondencias, formando con estas posiciones una trayectoria. Este programa por tanto, rastreaba las trayectorias de varios puntos de la secuencia de imágenes.

Sin embargo, este programa original no presentaba mucha robustez a la hora de hacer la asignación de correspondencias, haciéndola principalmente con una suma de diferencias absolutas, o SAD (*Sum of Absolute Differences*), de los descriptores de intensidad de color de cada punto (básicamente mide cuan semejantes son los escenarios que rodean a dos posibles correspondencias, si son muy similares se asigna una correspondencia).

El programa también carecía de un modo para evaluar la calidad de cada trayectoria rastreada, cosa que podría servir para indicar el momento justo en el cual hacer el cálculo del movimiento de la cámara, o para solo hacer este cálculo con ciertas trayectorias detectadas, aquellas que tuviesen calidad suficiente. Por estos problemas, se decidió crear un programa que pudiese solventarlos.

Se desarrolló un programa, optimización del programa original, que también detecta y rastrea las características presentes en las secuencias registradas por la cámara. Sin embargo, este programa realiza la detección de características y posterior búsqueda de correspondencias usando el descriptor ORB, y no el detector de características de Harris ni la SAD que el programa original usaba. El uso del descriptor ORB dotó al programa de una mejor calidad de trayectorias rastreadas, ocasionando muchas menos trayectorias erróneas, aunque generalmente el número de estas decayese respecto a las que decía detectar el primer programa.

Este programa también dota a cada trayectoria rastreada de un valor de calidad, calculado en base a los valores de detección y correspondencia dados por el descriptor ORB.

El programa también está dotado de una función filtro, que suaviza las trayectorias ya detectadas. El filtrado se realiza a partir de dos estimaciones, de la posible posición de la correspondencia en base a su velocidad, y de su posible posición detectada por el descriptor ORB. Este filtro también es capaz de hacer varios tipos de filtrados (dando distintos pesos a una estimación u otra) en función de la longevidad de la trayectoria.

Una muestra de las trayectorias detectadas por el programa creado en una secuencia de imágenes se puede observar en la Figura 2. Todos los detalles referentes al programa creado se pueden consultar en la Sección 3.3.

La elaboración de este programa fue extremadamente útil para entender gran parte de

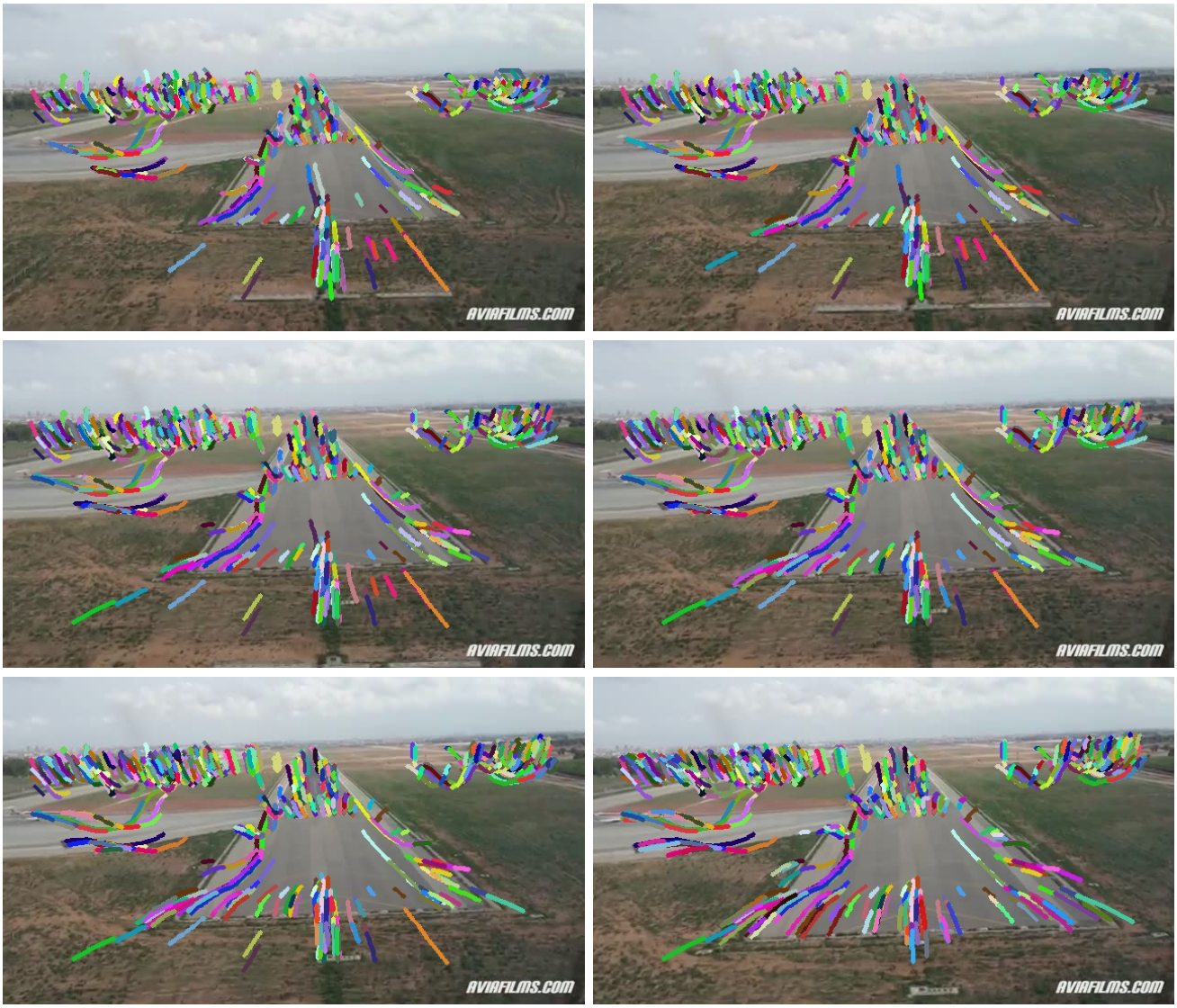


Figura 2: Trayectorias detectadas en una secuencia de imágenes mediante el programa realizado, en orden de aparición (de izquierda a derecha y de arriba abajo). Nótese como las trayectorias (que mantienen constante su color) intentan siempre apuntar al mismo punto en todas las imágenes.

lo leído en la documentación y para planificar parte de lo que hace falta realizar para el programa que lo usará en un futuro, aquel capaz de predecir la trayectoria de una cámara situada en una plataforma móvil.

Glosario

detección de características — Técnica que busca captar elementos en una imagen con el fin de poder identificarlos en otra, prediciendo así el movimiento de una imagen con respecto a otra. Es llamada también *feature detection*.

egomoción — Movimiento de una cámara con respecto a una referencia inmóvil.

IMU (*Inertial Measurement Unit* [unidad de medición inercial]) — Dispositivo capaz de medir la velocidad, rotación o las fuerzas específicas, como la gravedad, de un aparato.

lídar — Método para medir la distancia a un objetivo, iluminándolo con el haz de un láser pulsado.

nube de puntos — Conjunto de puntos designados por coordenadas 3D.

SLAM (*Simultaneous Localization and Mapping* [localización y mapeo simultáneos]) — Técnica en la que un agente construye un mapa de un entorno desconocido a la vez que mantiene conocida su propia localización.

odometría — Técnica para la estimación de la posición y orientación de un sistema móvil respecto a su entorno.

UAV (*Unmanned Aerial Vehicle* [vehículo aéreo no tripulado]) — Dron, aeronave sin conductor a bordo.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Memoria

1.1. Introducción

La odometría puede ser definida como la técnica que estima la posición y la orientación de un sistema móvil respecto a su entorno. Por lo tanto, la odometría visual es la técnica que estima esta posición y orientación de un sistema móvil mediante el análisis de secuencias de imágenes, vídeos, registradas por cámaras. Una definición más exacta de la odometría visual podía ser la sugerida en [2], que es la estimación de la posición de la cámara mediante la entrada visual, también llamada egomoción.

La odometría visual también puede usar información de otras fuentes como GPS, sensores de inercia, codificadores rotatorios, entre muchos otros, mejorando la calidad de sus predicciones.

Esta técnica también tiene numerosas aplicaciones, dentro de las cuales se destaca la ayuda que puede brindar a plataformas móviles para que conozcan su posición, lo que a su vez les puede servir para que se movilicen adecuadamente, algo muy útil para el desarrollo de la robótica. Por esto, la odometría visual es también útil para ayudar a ubicar a drones cuando estos se encuentran en entornos carentes de señal GPS; Para ayudar a los robots en su movilidad; Para ayudar a personas con discapacidades visuales, como sugerido en [3]; Para diseñar sistemas de asistencia para automóviles, como puede ser detectar un peligro en la carretera, detectar la presencia de un puesto de estacionamiento disponible o ayudar a que el vehículo se centre en el carril adecuado; Entre muchos otros diversos ejemplos.

El objetivo de este trabajo fue el de documentarse lo más posible sobre las distintas técnicas de odometría visual, y el de plasmar parte de lo aprendido, creando un programa capaz de rastrear las trayectorias recorridas por los elementos presentes en una secuencia de imágenes. Este programa servirá de base para otro que será capaz de estimar la posición de una cámara monocular móvil a partir de la grabación de esta, siguiendo su trayectoria.

Para documentarse, se buscaron distintos documentos en internet relacionados con la odometría visual, además de temas relacionados con ella, como ciertos algoritmos y

detectores de características.

La mayor parte de la documentación encontrada fue extraída del sitio web del proyecto KITTI Vision Benchmark Suite [1], [4], pues este sitio recoge referencias a distintos métodos de odometría visual. El Capítulo 2 de este documento está especialmente dedicado a comentar detalles destacables de los métodos referenciados en la página web del proyecto, intentando también reflejar el potencial de la odometría visual.

Para la creación del programa de rastreo de trayectorias en secuencias de imágenes, y para la creación de distintos programas de pruebas, se utilizó la librería de funciones de visión por computador OpenCV [5], y con ella se desarrollaron varios ficheros en lenguaje Python [6], utilizando también la librería de funciones para la manipulación de matrices NumPy [7]. El Capítulo 3 está dedicado a detallar el funcionamiento del programa creado, mencionando también algunos de los métodos que se valoraron para mejorar su eficacia. El Capítulo 4 muestra los resultados obtenidos y en el Capítulo 5 se detallan las conclusiones y futuras líneas de trabajo.

A continuación se describen parte de los conceptos básicos del procesado de imagen, y como con este se puede estimar la egomoción.

1.2. Base teórica

Para el procesado de imagen, una imagen 2D es una matriz cuadrada donde el mínimo elemento de esta es un punto, un píxel designado por dos coordenadas, x y y. Cada píxel, interno a la matriz que representa la imagen, refleja un color, que puede ser definido de distintas maneras. Por ejemplo, en la codificación RGB, un píxel es un vector de tres elementos, donde el primero designa el color rojo, el segundo el verde y el tercero el azul. El valor de cada elemento del vector designa la intensidad de cada color, y combinando los tres colores en distintas intensidades se pueden representar todos los colores posibles.

Otra codificación de colores muy útil para el procesado de imagen es la de escala de grises, donde cada píxel tiene un único valor que refleja la intensidad, en niveles de gris, que este tiene. Esta codificación es especialmente útil para aliviar el computo matemático con imágenes, entre otras ventajas.

Debido a que las imágenes son matrices, la mayoría de los cálculos relacionados con ellas son cálculos matriciales. En los siguientes apartados se describen algunos conceptos fundamentales para entender como se puede reconstruir el camino recorrido por una cámara a partir de las imágenes grabadas por esta. Parte de la información descrita se puede encontrar en [8]. La notación en este documento representa a los vectores en negrita.

1.2.1. Matriz de rotación

Una rotación puede definirse como un desplazamiento circular en cualquier eje. Matemáticamente, la rotación de un punto se puede expresar como la multiplicación de una matriz de rotación R por ese punto. En 3D R es un producto de tres matrices de rotación: R_z, R_y y R_x, en ese orden, donde cada una de estas tres designan a una rotación alrededor de un eje x, y o z, con un ángulo θ, β o α respectivamente. En la Ecuación 1.1 se puede ver la definición de la matriz de rotación pura y en la Ecuación 1.2 las de las tres matrices que la componen. En la Figura 1.1 se pueden observar los tres ángulos de la rotación en 3D que son representados por las matrices de rotación.

$$R = R_z \cdot R_y \cdot R_x \tag{1.1}$$

$$\begin{gathered} R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ R_y = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \end{gathered} \tag{1.2}$$

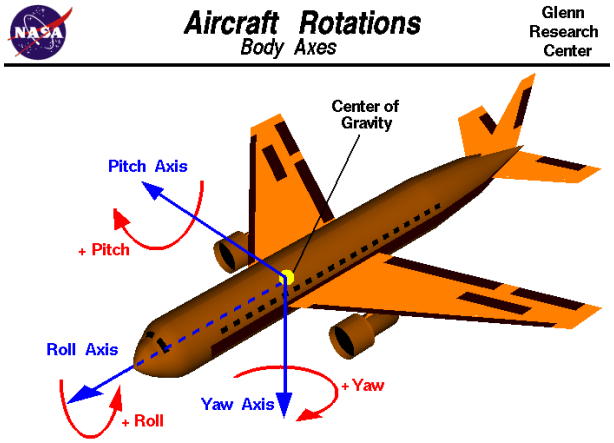


Figura 1.1: Ángulos de giro en 3D. Cada uno de los ejes coordenados está representado por una flecha azul y el giro alrededor de dicho eje por su flecha roja más cercana. Imagen extraída de [9].

1.2.2. Vector de traslación

Una traslación puede definirse como un desplazamiento lineal en cualquier eje coordenado (como en cualquiera de las flechas azules de la Figura 1.1). Matemáticamente, la traslación de un punto se puede expresar como la multiplicación de un vector de traslación $\boldsymbol{t}$ por

dicho punto. En 3D $\boldsymbol{t}$ designa un desplazamiento t_x en el eje x, t_y en el eje y y t_z en el eje z. Este vector se puede observar en la Ecuación 1.3.

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{1.3}$$

1.2.3. Matriz de parámetros intrínsecos de la cámara

Para un sistema capaz de hacer rastrear la trayectoria de una cámara móvil, es necesario modelar matemáticamente la cámara usada. Un modelado de cámara generalizable a otras cámaras es el de la cámara *pinhole* o cámara estenopeica, que se representaría con la matriz K que puede verse en la Ecuación 1.4. En esta matriz, f representaría la distancia focal de la cámara, s el parámetro *skew* (que definiría un cierto aplastamiento en la imagen), mientras que p_x y p_y representan el llamado punto principal, que suele coincidir con el centro de la imagen en píxeles.

$$K = \begin{bmatrix} f & s & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.4}$$

Para obtener los parámetros que definen a K es necesario calibrar la cámara, aplicando uno de los distintos métodos de calibración disponibles.

1.2.4. Matriz de proyección de la cámara

Una cámara sirve para convertir un punto 3D $\boldsymbol{P}$ real a un punto 2D $\boldsymbol{p}$, mediante una matriz de proyección de la cámara T. Esta matriz tiene en cuenta la matriz de parámetros intrínsecos de la cámara, K, además de los parámetros extrínsecos, los relativos a su posición a la hora de tomar la foto, representados por las matrices R y $[I|\boldsymbol{t}]$ (siendo I la matriz identidad). En la Ecuación 1.5 se aprecia el contenido de la matriz T y la relación entre el punto $\boldsymbol{P}$ y el punto $\boldsymbol{p}$.

$$\boldsymbol{p} = T \cdot \boldsymbol{P} = K \cdot R \cdot [I|\boldsymbol{t}] \cdot \boldsymbol{P} \tag{1.5}$$

Para intentar predecir el movimiento de la cámara es necesario conocer los parámetros extrínsecos de la cámara, aquellos que describen su posición en el momento de tomar la foto, R y $\boldsymbol{t}$. El intentar dar solución al problema de la obtención de estos parámetros, problema PnP (*Perspective-n-Point*), es llamado estimación de pose.

1.2.5. Estimación de pose

Al sacar una foto, solamente son conocidos los parámetros K (se asume que se seguirá utilizando siempre una misma cámara en las mismas condiciones, sin variar la distancia focal) y $\boldsymbol{p}$ (puntos en la foto), teniéndose de incógnitas a $\boldsymbol{P}$ (puntos 3D reales equivalentes a los de la foto $\boldsymbol{p}$) y a los parámetros extrínsecos R y $\boldsymbol{t}$, y. Los parámetros extrínsecos

pueden ser medidos con métodos mecánicos (como con una cinta métrica y un medidor de ángulos), pero se pueden intentar averiguar a partir fotos tomadas por la cámara.

Para estimar estos parámetros extrínsecos existen distintos algoritmos, que se aprovechan de tener al menos dos fotografías en las que estén presentes los mismos puntos 3D $\boldsymbol{P}$ (por ejemplo, que en las dos fotos se vea la misma esquina), desde distintas posición y ángulo, dando lugar a distintos puntos equivalentes $\boldsymbol{p}$ (cada $\boldsymbol{P}$ daría lugar a dos $\boldsymbol{p}$, uno en cada fotografía). La Figura 1.2 refleja esta idea, un mismo punto 3D rojo es proyectado en dos imágenes de manera distinta. Estos puntos, $\boldsymbol{p}$, proyectados a partir de un mismo punto 3D son llamados correspondientes.

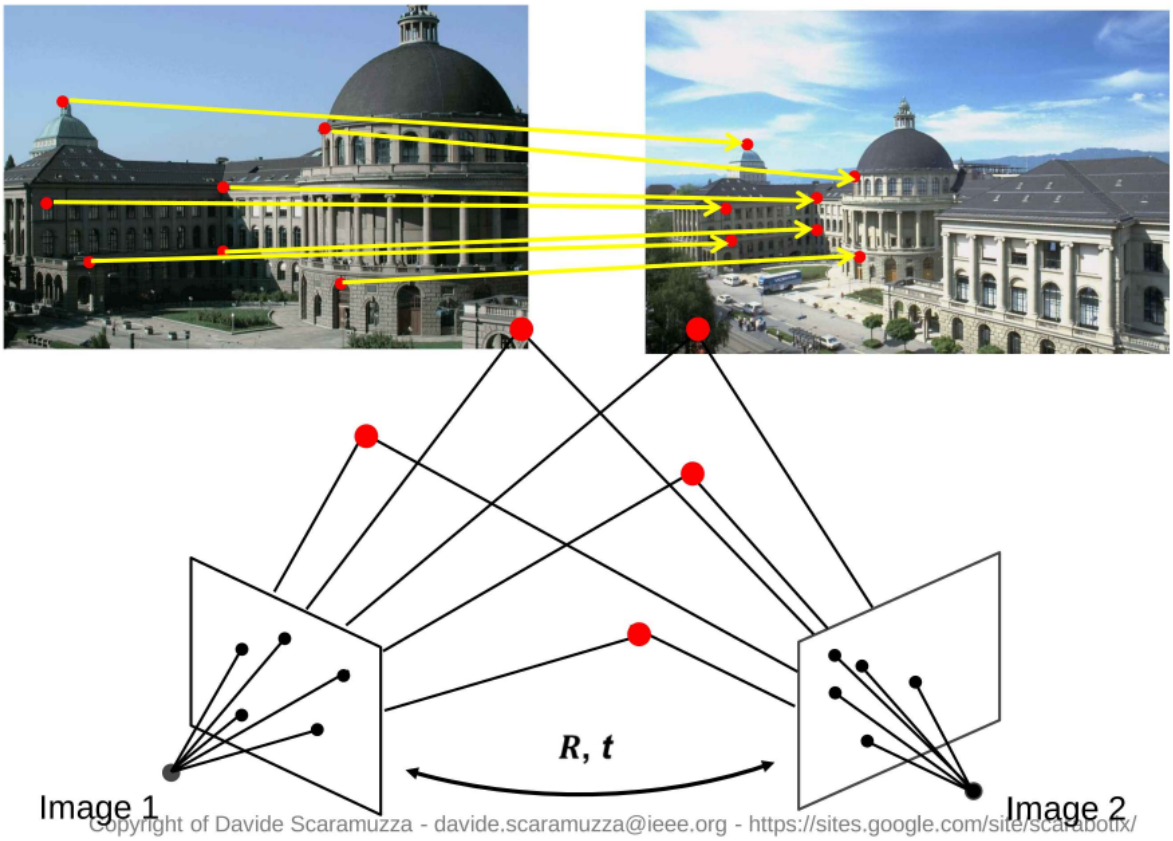


Figura 1.2: Puntos 3D (coloreados de rojo) proyectados en dos imágenes distintas. Los puntos unidos por las líneas amarillas son los mismos, pero desde distinta perspectiva. Las líneas negras muestran como un mismo punto es proyectado de distinta manera en cada imagen, siendo estas proyecciones puntos correspondientes. Imagen extraída de [8].

Al tener varias correspondencias (por ejemplo unas cuatro al usar el algoritmo POSIT [10]), se es capaz de estimar una matriz de pose Rt, descrita en la Ecuación 1.6, de la cual se podrán extraer la matriz R y el vector $\boldsymbol{t}$.

$$Rt = R \cdot [I| - \boldsymbol{t}] = [R| - R \cdot \boldsymbol{t}] \tag{1.6}$$

Para realizar esta estimación de pose se destaca el algoritmo POSIT, del cual se puede encontrar un código con su funcionamiento en [11], y otros algoritmos que presentan

mejoras frente a este, como el algoritmo EPnP [12] y el DLT (*Direct Linear Transform*), entre otros.

1.2.6. Epipolos y foco de expansión

Considerando un escenario visualizado por dos cámaras desde dos puntos de vista distintos (o la misma cámara en dos posiciones distintas), un epipolo es la proyección del centro de proyección de una primera cámara (por ejemplo, si el foco fuese nulo el centro de proyección coincidiría con la posición de la cámara) sobre el plano imagen de la segunda (es como si se proyectase la primera cámara sobre la imagen registrada por la segunda). La Figura 1.3 muestra los epipolos sobre los planos imagen de dos cámaras.

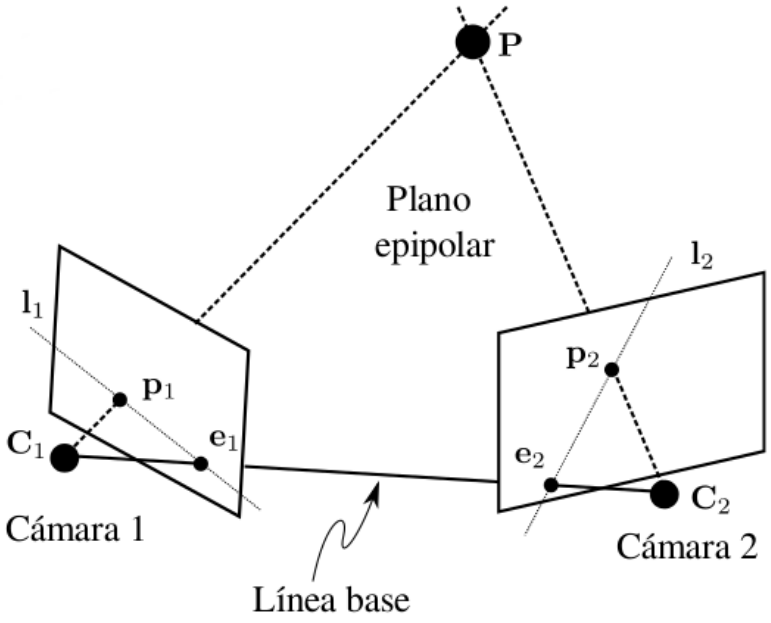


Figura 1.3: Epipolos y líneas epipolares. Los epipolos están designados por la letra e, las líneas epipolares por l. Un punto 3D es llamado P, las proyecciones de este punto sobre los planos imagen de las cámaras están designadas por p y las cámaras por C.

Cuando se usa una única cámara que tomó dos fotos a un escenario desde posiciones distintas, el epipolo visto desde la segunda foto mostrará la dirección a la que se desplazó la cámara. Este epipolo es llamado foco de expansión [13], y por su misma definición puede ser utilizado para predecir el movimiento de la cámara. Una representación del foco de expansión en una imagen se puede apreciar en la Figura 1.4.

1.2.7. Matriz fundamental

Para hallar el epipolo se necesita encontrar la intersección entre líneas epipolares, líneas que pasan por el epipolo y por un punto 2D $\boldsymbol{p}$ equivalente de uno 3D $\boldsymbol{P}$. Estas líneas $\boldsymbol{l'}$, presentes en la Figura 1.3, cumplen la Ecuación 1.7, donde F es la matriz fundamental, que su a vez cumple la Ecuación 1.8, donde $\boldsymbol{p}$ es el punto equivalente 2D de uno 3D $\boldsymbol{P}$ en

Figura 1.4: Foco de expansión en una carretera, las líneas epipolares convergen en el punto hacía el cual se movió la cámara. Imagen extraída de [13].

la primera cámara (o primera posición), mientras que $\boldsymbol{p'}^{\intercal}$ es la transpuesta del equivalente 2D del mismo punto 3D $\boldsymbol{P}$ pero en la segunda cámara (o segunda posición).

$$\boldsymbol{l'} = F \cdot \boldsymbol{p} \tag{1.7}$$

$$\boldsymbol{p'}^{\intercal} \cdot F \cdot \boldsymbol{p} = 0 \tag{1.8}$$

Un algoritmo como DLT permite que al conseguir varias equivalencias de puntos 2D entre dos imágenes se pueda hallar la matriz F, por tanto las líneas epipolares, y con ella el foco de expansión, permitiendo también, si se dispone de la matriz K, hallar la llamada matriz esencial E, y con esta estimar los parámetros extrínsecos de la cámara.

1.2.8. Matriz esencial

La matriz esencial E se puede hallar a partir de la matriz F y la matriz K según lo descrito en la Ecuación 1.9.

$$E = K^{\intercal} \cdot F \cdot K \tag{1.9}$$

A partir de esta matriz esencial y por medio de la descomposición en valores singulares de la matriz, se pueden obtener estimaciones de la matriz de rotación y del vector de rotación deseados, pues la matriz esencial cumple la Ecuación 1.10, donde R es la matriz ortogonal de rotación, t el vector de traslación y $[]_x$ se corresponde con la matriz antisimétrica.

$$E = [t]_x \cdot R \tag{1.10}$$

La descomposición da lugar a cuatro posibles soluciones, pues estima dos posibles matrices de rotación, R_1 y R_2, y dos vectores de rotación, $\boldsymbol{t}$ y $-\boldsymbol{t}$. Una de las combinaciones de la matriz y el vector corresponden al escenario correcto, otra a este mismo escenario

invertido 180° sobre la línea base de la cámara, y las dos restantes a las reflexiones de estas dos anteriores. Se puede deducir cual es la combinación de matriz de rotación y vector de traslación correcta sabiendo que un punto del escenario captado por la cámara debe estar en frente de esta (y por tanto no en una posición reflejada o bajo su línea base). Por ello, conociendo un único punto captado por la cámara se pueden obtener finalmente la correcta matriz R y el vector $-\boldsymbol{t}$, como se describe en [14].

1.2.9. Estimación del desplazamiento

Habiendo obtenido correctamente los parámetros extrínsecos R y $\boldsymbol{t}$ ya es posible estimar el desplazamiento entre una foto y una posterior. Suponiendo que los parámetros fueron obtenidos por medio de una primera foto sacada en el instante $k-1$ y una posterior en el instante k, los parámetros obtenidos se pueden renombrar como R_k y $\boldsymbol{t}_k$, respectivamente, dando lugar a una matriz de pose Rt_k.

Esta matriz de pose solamente contiene información sobre el desplazamiento realizado entre los fotogramas en los que fue calculada (asociados al instante k). Para encontrar la posición exacta de la cámara se debe considerar la pose en los instantes anteriores (si los hubo). Por motivos matemáticos (para que pueda ser correctamente multiplicada), a la matriz de pose se le debe añadir una cuarta fila, que contenga al vector $[0, 0, 0, 1]$, volviéndola una matriz cuadrada equivalente. Tras añadir esta fila podrá ser multiplicada como indica la Ecuación 1.11, donde M_{k-1} es la posición de la cámara en el instante anterior. Este cálculo permitirá hallar la posición actual de la cámara, M_k, y servirá a su vez para hallar la posición de la cámara en un instante posterior $k+1$, si se hallase una matriz de pose para ese instante.

$$M_k = Rt_k \cdot M_{k-1} \tag{1.11}$$

El único problema adicional a resolver es el de realizar una estimación de la distancia recorrida. Sin conocer esta distancia no es posible escalar la posición de la cámara a una posición del mundo real. Para hallar la distancia existen diversos métodos: Con el uso de ayudas externas se podría conocer la distancia directamente, pero si no es posible se podría estimar a partir de la velocidad medida por una IMU (Unidad de Medición Inercial), estimarla a partir de la distancia entre los focos si se usó una cámara estéreo, o estimarla a partir de la altura de la cámara; Sin ayudas externas es necesario hacer estimaciones de la profundidad de las fotos, como podría ser usando campos aleatorios de Markov, como en [15], o el algoritmo de Levenberg-Marquardt, como utilizaron en el método V-LOAM [16]; Entre otras maneras.

1.3. Sumario general

Como se verá en el Capítulo 2, hay una gran cantidad de métodos, y dentro de estos muchos algoritmos y funciones distintas, que pueden ser de gran utilidad para la odometría visual. Se concluyó que todavía es necesario desarrollar métodos de odometría visual monocular que mantengan un nivel de efectividad en tiempo real más cercano a

los métodos de odometría lídar y odometría visual estéreo, y que gracias a la relación entre estos tres tipos de odometrías, este desarrollo puede realizarse en conjunto.

Al crear el programa descrito en el Capítulo 3, para rastrear las características en una secuencia de imágenes, se tuvo la oportunidad de utilizar algunas de las ecuaciones, detectores y filtros presentados en el Capítulo 2, confirmando su efectividad.

Para continuar y perfeccionar en gran medida el programa realizado en este proyecto, se debe crear un programa que estime la trayectoria recorrida por la cámara. Para hacer esta estimación, este nuevo programa debería utilizar el programa ya creado, realizando el cálculo de los parámetros extrínsecos en función de la calidad de las trayectorias detectadas. Para ello, y posiblemente lo más complicado de implementar, es también necesario realizar una estimación de la distancia recorrida entre cada captura de la cámara, para que el mapa predicho este correctamente escalado. Esta estimación de la distancia también podría usarse para mejorar el programa creado en este proyecto, mejorando el filtrado que realiza en base a la velocidad de las características.

Además, como dicho en el Capítulo 5, sería práctico crear un programa de aprendizaje automatizado para que este encuentre los parámetros ideales a usar en el programa de predicción de trayectorias creado. La enorme cantidad de parámetros vuelve muy difícil para un humano encontrar la mejor combinación a base de probar repetidamente distintos valores.

Capítulo 2

Métodos de odometría visual

Con el propósito de informarse sobre distintos mecanismos e utilidades de la odometría visual se leyeron varios artículos sobre el tema, y en este capítulo se citarán algunos, dando pequeñas explicaciones sobre ellos y su relevancia.

Un gran referente usado para obtener bibliografía sobre odometría es el proyecto KITTI Vision Benchmark Suite [1], [4]. Los integrantes del proyecto dotaron a un automóvil con dos cámaras de alta resolución, a color y en escala de grises, con un escáner láser de Velodyne y con un sistema de localización GPS, para que el vehículo recopilase distintos datos al circular por la ciudad de Karsruhe, en áreas rurales y autopistas.

Los datos recopilados incluyen 22 secuencias de vídeo en visión estereoscópica, guardadas en formato png, que se pueden descargar libremente en la sección de odometría de la página web del proyecto[1]. Estas secuencias se pueden utilizar para intentar predecir, mediante odometría visual, las trayectorias recorridas por el automóvil.

Esta página también incluye una clasificación que recopila distintos métodos usados por distintas personas para intentar predecir dichas trayectorias. Para medir la eficacia de cada método, estos aportan datos del error de traslación (en porcentaje) y de rotación (en grados por metro) medios. Estos errores medios son medidos respecto a la referencia medida por el vehículo, siguiendo la Ecuación 2.1 y la Ecuación 2.2 (extraidas de [4]), donde F es un conjunto de fotogramas (i, j), $\hat{p}$ es la pose (lo que determina su posición en un determinado momento) estimada de la cámara, p la pose real de la cámara, $\ominus$ es el operador composicional inverso [17] y $\angle[\,]$ es el ángulo de rotación. Un extracto de esta clasificación del KITTI se puede observar en la Tabla 2.1.

$$E_{rot}(F) = \frac{1}{F} \sum_{(i,j)\in F} \angle[(\hat{\boldsymbol{p}}_{\boldsymbol{j}} \ominus \hat{\boldsymbol{p}}_{\boldsymbol{i}}) \ominus (\boldsymbol{p}_{\boldsymbol{j}} \ominus \boldsymbol{p}_{\boldsymbol{i}})] \tag{2.1}$$

$$E_{rot}(F) = \frac{1}{F} \sum_{(i,j)\in F} ||(\hat{\boldsymbol{p}}_{\boldsymbol{j}} \ominus \hat{\boldsymbol{p}}_{\boldsymbol{i}}) \ominus (\boldsymbol{p}_{\boldsymbol{j}} \ominus \boldsymbol{p}_{\boldsymbol{i}})||_2 \tag{2.2}$$

[1] http://www.cvlibs.net/datasets/kitti/eval_odometry.php [Accedido: 10 de marzo de 2019]

Tabla 2.1: Nombre de los métodos usados para reproducir las trayectorias registradas por el proyecto KITTI [1], sus errores de traslación y rotación, y sus tiempos de ejecución. Actualizada a día 1 de julio de 2019.

Método	**Traslación**	**Rotación**	**Tiempo de ejecución**
RLO [18]	0 %	0 [deg·m^{-1}]	0.05 s / GPU
V-LOAM [16]	0.56 %	0.0013 [deg·m^{-1}]	0.1 s / 2 núcleos
LOAM [19]	0.59 %	0.0014 [deg·m^{-1}]	0.1 s / 2 núcleos
SOFT2 [20]	0.65 %	0.0014 [deg·m^{-1}]	0.1 s / 2 núcleos
IMLS-SLAM [21]	0.69 %	0.0018 [deg·m^{-1}]	1.25 s / 1 núcleo
MC2SLAM [22]	0.69 %	0.0016 [deg·m^{-1}]	0.1 s / 4 núcleos
LG-SLAM [23]	0.82 %	0.0020 [deg·m^{-1}]	0.2 s / 4 núcleos
RotRocc+ [24], [25], [26]	0.83 %	0.0026 [deg·m^{-1}]	0.25 s / 2 núcleos
LIMO2_GP [27]	0.84 %	0.0022 [deg·m^{-1}]	0.2 s / 2 núcleos
GDVO [28]	0.86 %	0.0031 [deg·m^{-1}]	0.09 s / 1 núcleo
LIMO2 [27]	0.86 %	0.0022 [deg·m^{-1}]	0.2 s / 2 núcleos
CPFG-slam [29]	0.87 %	0.0025 [deg·m^{-1}]	0.03 s / 4 núcleos
SOFT [30]	0.88 %	0.0022 [deg·m^{-1}]	0.1 s / 2 núcleos
RotRocc [24]	0.88 %	0.0025 [deg·m^{-1}]	0.3 s / 2 núcleos
DVSO [31]	0.90 %	0.0021 [deg·m^{-1}]	0.1 s / GPU
LIMO [32]	0.93 %	0.0026 [deg·m^{-1}]	0.2 s / 2 núcleos
Stereo DSO [33]	0.93 %	0.0020 [deg·m^{-1}]	0.1 s / 1 núcleo
Elbrus [34]	0.98 %	0.0023 [deg·m^{-1}]	0.1 s / 1 núcleo
ROCC [35]	0.98 %	0.0028 [deg·m^{-1}]	0.3 s / 2 núcleos
cv4xv1-sc [36]	1.09 %	0.0029 [deg·m^{-1}]	0.145 s / GPU
VINS-Fusion [37]	1.09 %	0.0033 [deg·m^{-1}]	0.1s / 1 núcleo
MonoROCC [38]	1.11 %	0.0028 [deg·m^{-1}]	1 s / 2 núcleos
DEMO [39]	1.14 %	0.0049 [deg·m^{-1}]	0.1 s / 2 núcleos
ORB-SLAM2 [40]	1.15 %	0.0027 [deg·m^{-1}]	0.06 s / 2 núcleos
ElbrusFast [41]	1.15 %	0.0032 [deg·m^{-1}]	0.018 s / 1.5 núcleos
NOTF [42]	1.17 %	0.0035 [deg·m^{-1}]	0.45 s / 1 núcleo
S-PTAM [43], [44]	1.19 %	0.0025 [deg·m^{-1}]	0.03 s / 4 núcleos
S-LSD-SLAM [45]	1.20 %	0.0033 [deg·m^{-1}]	0.07 s / 1 núcleo
VoBa [46]	1.22 %	0.0029 [deg·m^{-1}]	0.1 s / 1 núcleo
STEAM-L WNOJ [47]	1.22 %	0.0058 [deg·m^{-1}]	0.2 s / 1 núcleo
LiViOdo [48]	1.22 %	0.0042 [deg·m^{-1}]	0.5 s / 1 núcleo
SLUP [49]	1.25 %	0.0041 [deg·m^{-1}]	0.17 s / 4 núcleos
STEAM-L [50]	1.26 %	0.0061 [deg·m^{-1}]	0.2 s / 1 núcleo
FRVO [51]	1.26 %	0.0038 [deg·m^{-1}]	0.03 s / 1 núcleo
MFI [52]	1.30 %	0.0030 [deg·m^{-1}]	0.1 s / 1 núcleo
TLBBA [53]	1.36 %	0.0038 [deg·m^{-1}]	0.1 s / 1 núcleo
2FO-CC [54]	1.37 %	0.0035 [deg·m^{-1}]	0.1 s / 1 núcleo
SuMa [55]	1.39 %	0.0034 [deg·m^{-1}]	0.1 s / 1 núcleo
ProSLAM [56]	1.39 %	0.0035 [deg·m^{-1}]	0.02 s / 1 núcleo
StereoSFM [57]	1.51 %	0.0042 [deg·m^{-1}]	0.02 s / 2 núcleos

SSLAM [58], [59], [60]	1.57 %	0.0044 [deg·m^{-1}]	0.5 s / 8 núcleos
eVO [61]	1.76 %	0.0036 [deg·m^{-1}]	0.05 s / 2 núcleos
Stereo DWO [62]	1.76 %	0.0026 [deg·m^{-1}]	0.1 s / 4 núcleos
BVO [63]	1.76 %	0.0036 [deg·m^{-1}]	0.1 s / 1 núcleo
D6DVO [64], [65]	2.04 %	0.0051 [deg·m^{-1}]	0.03 s / 1 núcleo
PMO / PbT-M2 [66]	2.05 %	0.0051 [deg·m^{-1}]	1 s / 1 núcleo
SSLAM-HR [58], [59], [60]	2.14 %	0.0059 [deg·m^{-1}]	0.5 s / 8 núcleos
FTMVO [67]	2.24 %	0.0049 [deg·m^{-1}]	0.11 s / 1 núcleo
PbT-M1 [68], [69]	2.38 %	0.0053 [deg·m^{-1}]	1 s / 1 núcleo
VISO2-S [70]	2.44 %	0.0114 [deg·m^{-1}]	0.05 s / 1 núcleo
MLM-SFM [71], [72]	2.54 %	0.0057 [deg·m^{-1}]	0.03 s / 5 núcleos
GT_VO3pt [73]	2.54 %	0.0078 [deg·m^{-1}]	1.26 s / 1 núcleo
RMCPE+GP [74]	2.55 %	0.0086 [deg·m^{-1}]	0.39 s / 1 núcleo
VO3pt [3], [75]	2.69 %	0.0068 [deg·m^{-1}]	0.56 s / 1 núcleo
TGVO [76]	2.94 %	0.0077 [deg·m^{-1}]	0.06 s / 1 núcleo
VO3ptLBA [3], [75]	3.13 %	0.0104 [deg·m^{-1}]	0.57 s / 1 núcleo
PLSVO [77]	3.26 %	0.0095 [deg·m^{-1}]	0.20 s / 2 núcleos
BLF [78]	3.49 %	0.0128 [deg·m^{-1}]	0.7 s / 1 núcleo
CFORB [79]	3.73 %	0.0107 [deg·m^{-1}]	0.9 s / 8 núcleos
VOFS [80], [81]	3.94 %	0.0099 [deg·m^{-1}]	0.51 s / 1 núcleo
VOFSLBA [80], [81]	4.17 %	0.0112 [deg·m^{-1}]	0.52 s / 1 núcleo
CUDA-EgoMotion [82]	4.36 %	0.0052 [deg·m^{-1}]	0.001 s / GPU
BCC [78]	4.59 %	0.0175 [deg·m^{-1}]	1 s / 1 núcleo
EB3DTE+RJMCM [83], [84]	5.45 %	0.0274 [deg·m^{-1}]	1 s / 1 núcleo
VISO2-M + GP [70], [71]	7.46 %	0.0245 [deg·m^{-1}]	0.15 s / 1 núcleo
BLO [78]	9.21 %	0.0163 [deg·m^{-1}]	0.1 s / 1 núcleo
VISO2-M [70]	11.94 %	0.0234 [deg·m^{-1}]	0.1 s / 1 núcleo
OABA [85]	20.95 %	0.0135 [deg·m^{-1}]	0.5 s / 1 núcleo

2.1. Métodos en el proyecto KITTI

Se citarán a continuación parte de los métodos presentes en la clasificación del proyecto KITTI, mencionando los más interesantes detalles de varios de estos. Cabe resaltar que muchos de estos están enfocados para realizar la técnica SLAM (*Simultaneous Localization and Mapping* [localización y mapeo simultáneos]), en la cual un agente intenta construir un mapa de su entorno mientras mantiene conocida su posición. Los métodos citados están separados en los tres tipos principales de odometrías usadas: lídar, estéreo y monocular.

2.1.1. Métodos con odometría lídar

Se puede definir la odometría lídar como la técnica para la estimación de la posición y orientación de un sistema móvil respecto a su entorno, siendo un lídar un dispositivo que permite medir la distancia a un objetivo, iluminando este con un láser de pulsos. Para calcular la distancia se mide el tiempo en el que este láser rebota de vuelta al emisor. Con

estas estimaciones de distancia se crean nubes de puntos, puntos en el espacio 3D, que representan el espacio en el que se encuentra el sistema móvil.

De los varios métodos publicados en el proyecto KITTI, aquellos que utilizan estas nubes de puntos poseen, en general, los menores errores de rotación.

Estrictamente hablando la odometría lídar no es odometría visual, al no necesitar secuencias de imágenes. Pese a ello, se decidió incluirla en este documento, por su presencia en el proyecto KITTI, su potencial y la capacidad de ser combinada con la odometría visual pura.

Un método a destacar es RLO [18], que (a día 1 de julio de 2019) está clasificado como el mejor método de odometría en la base de datos del KITTI, teniendo los menores errores medios de traslación y de rotación. Según la página web del KITTI, estos errores serían nulos, volviéndolo un método perfecto, al menos para las secuencias del proyecto. Según el propio documento, quizás desactualizado, que describe el método, este tendría un error de traslación del 0.57 % y de rotación de 0.0013 $[\text{deg}\cdot\text{m}^{-1}]$.

Existe la posibilidad de que los errores descritos por la página del proyecto KITTI sean erróneos, visto que todavía no están disponibles los mapas de trayectorias generados por este método, y por lo tanto puede que los descritos en el propio documento sean más exactos. Tampoco resulta inusual que en el documento que describe al método se mencionen errores mayores de los presentes en la página del proyecto, sucede con varios otros métodos en el proyecto, dando a entender que estos fueron mejorados desde su publicación.

El método RLO intenta mapear el entorno del vehículo que graba las imágenes mediante mapas de cuadrículas de ocupación, mapas que definen el entorno del vehículo mediante un conjunto de cuadrículas (celdas), que pueden ser definidas como ocupadas por un objeto o no ocupadas. Para determinar si una celda del mapa está o no ocupada el método utiliza una fórmula que a su vez usa la inferencia bayesiana.

Para la estimación de pose (orientación y posición del vehículo) se asume que no hay variación de altura, y se hace una estimación en el cambio de la rotación y de la traslación de las características del mapa creado (este mapa gira al girar el vehículo) por medio de la técnica POC (*Phase-Only Correlation*) [86].

Otro método a destacar es el V-LOAM (Visual-lidar Odometry and Mapping [odometría y mapeo visual-lídar]) [16], una combinación de técnicas de odometría visual y de odometría lídar, basado en el método LOAM [19]. Actualmente (a día 1 de julio de 2019) está clasificado como el segundo mejor método de odometría en la base de datos del KITTI, teniendo los segundos menores errores medios de traslación y de rotación, 0.57 % y 0.0013 $[\text{deg}\cdot\text{m}^{-1}]$ respectivamente.

Se menciona que los mapas creados con este método tanto como en ambientes de interior y de exterior son suficientemente precisos sin necesidad de postprocesamiento, además de presentar robustez frente a falta de iluminación. Los resultados de algunos experimentos realizados con el método se pueden consultar en un vídeo de libre acceso.[2]

[2] https://www.youtube.com/watch?v=-6cwhPMAap8 [Accedido: 10 de marzo de 2019]

El método usa la odometría visual para estimar la egomoción y para registrar nubes de puntos, y usa la odometría lídar para refinar los datos previamente recolectados por la primera. Los procesos no operan a la misma velocidad, el proceso que usa la odometría visual se realiza a una mayor frecuencia, a la misma que los fotogramas por segundo (60 Hz), y la lídar a una menor (1 Hz), la misma velocidad a la que realiza un barrido.

Para la odometría visual, resultó interesante la manera que se utilizó para estimar la distancia entre dos capturas de la cámara, cuando la distancia no es conocida la estiman presentando un sistema de ecuaciones obtenidas a partir de la función de estimación de movimiento, que resuelven usando el algoritmo de Levenberg-Marquardt, dando mayor valor a algunas características detectadas que a otras y buscando iterativamente la estimación de movimiento óptima.

Otro método que usa la odometría lídar en conjunto con la visual monocular es el LIMO [32], que utilizá la parte visual para iniciar el llamado *bundle adjustment*, o ajuste de haces, un método para intentar realizar VSLAM (*Visual SLAM* [SLAM visual]). En el documento dedicado al método también se explica como se hace una estimación de la profundidad con grupos de nubes de puntos detectados por el lídar, y además también se comenta que los lídares no tienen tanta resolución vertical como la tienen horizontal. Además, los autores dan libre acceso a los códigos usados para realizar el método.[3]

En el artículo sobre el método DEMO [39] comentan que asocian la profundidad de las características detectadas visualmente con aquellas detectadas con el lídar. Mencionan también la existencia de una librería de código abierto iSAM [87] que usan para realizar el ajuste de haces que a su vez refina el movimiento predicho entre dos fotos. Su método de rastreo de características usa el detector de Harris [88] con el método KLT (Kanade-Lucas-Tomasi) [89], [90].

Otro método de odometría lídar es el IMLS-SLAM [21], en cuyo artículo se menciona que se borra del escaneado lídar los objetos dinámicos, cosa que hacen eliminando los objetos con un tamaño suficiente para creer que lo sean. También se han de nombrar los métodos STEAM-L WNOJ [47] y STEAM-L [50], que estiman la trayectoria mediante la técnica lídar STEAM (*Simultaneous Trajectory Estimation and Mapping* [mapeo y estimación de trayectorias simultaneos]) [91], y el método SuMa [55].

2.1.2. Métodos con odometría estéreo

La odometría estéreo es un tipo de odometría visual, con la distinción de que utiliza cámaras estereoscópicas, que al usar dos objetivos pueden crear una imagen 3D, simulando la visión humana. Una de sus mayores desventajas es su dependencia de una calibración de parámetros extrínsecos precisa, como mencionan en [32], .

Un método a destacar es el SOFT2 (*Stereo Odometry and Feature Tracking 2* [odometría estéreo y rastreo de características 2]) [20], con un algoritmo basado en el rastreo de

[3] https://github.com/johannes-graeter/limo [Accedido: 10 de marzo de 2019]

características captadas por las dos cámaras propias de la odometría estéreo. Actualmente (a día 1 de julio de 2019) está clasificado como el mejor método de odometría estéreo y el quinto mejor de odometría en la base de datos del KITTI, teniendo un error medio de traslación del 0.65 % y un error medio de rotación del 0.0014 [deg·m^{-1}]. Su método esta principalmente enfocado para su uso en los UAV (*Unmanned Aerial Vehicle* [vehículo aéreo no tripulado]) y para que funcione en estos en tiempo real. Hacen uso de una IMU (*Inertial Measurement Unit* [unidad de medición inercial]) interna al dron, dispositivo capaz de medir velocidad, rotación y fuerzas específicas de este, aunque como el proyecto KITTI no proporciona datos de este tipo, puede resultar indiferente. El método es similar a ORB-SLAM2 [40] (usando también el descriptor ORB [92]) pero se diferencia en que usa odometría visual SOFT [30], además de dar resultados deterministas (en cada ejecución siempre dará la mismos resultados) en el mapeo SLAM.

El método realiza dos procesos en paralelo, y no secuenciales el V-LOAM, uno de odometría visual y otro de mapeo SLAM, que fusionan su información tras sus ejecuciones. El IMU no es indispensable, pero ayuda a estimar la pose, seleccionando un radio para el rastreo de características en cada imagen.

Las características son detectadas con reconocimiento de regiones y de esquinas. Para buscar las correspondencias entre imágenes se usa el mismo procedimiento propuesto en Geiger et al., 2011 [70], para métodos estéreo: Considerando dos intervalos de tiempo, uno actual y uno anterior, cada uno con dos imágenes izquierdas y derechas, se examinan las características en la imagen izquierda actual, luego se compararan con la imagen izquierda anterior, luego con la derecha anterior, luego en la derecha actual y de nuevo en la imagen izquierda actual; Si la característica fue correctamente rastreada en todas las imágenes, se considera válida.

A la hora de elegir las características se aseguran que estas estén uniformemente distribuidas en la imagen, como también realizaron Kiti et al., 2010 [76], y se van quedando con las características consideradas más fuertes, pero asegurándose de dejar entrar también a las más nuevas aunque no sean tan fuertes, procedimiento realizado en el método SOFT [30]. Los puntos analizados son después filtrados con el algoritmo RANSAC [93], haciendo que este escoja entre 1 y 5 puntos aleatorios según el entorno del vídeo usado (con las secuencias del proyecto KITTI usaron 1 punto). También utilizan un filtro de Kalman [94] para refinar la estimación de rotación del giroscopio de la IMU.

Cabe destacar una alternativa al algoritmo RANSAC apodada MASOR, contenida en el método ROCC [35], RotRocc [25] y RotRocc+ [26], que en lugar de coger un mínimo de puntos aleatorios del conjunto de detectados, eligen el máximo para intentar hacer un cálculo del movimiento. Los métodos dieron resultados con errores de traslación inferiores al 1 %.

En el artículo dedicado a GDVO [28] se comenta que el rastreo de características está limitado, no pudiendo usar todo el contenido de la imagen y siendo incapaz de hacer una reconstrucción 3D completa, para lo que sugieren un método de odometría visual estéreo directa usando una optimización mediante jacobianos.

El método StereoDSO [33] realiza un rastreo de características usando alineamiento directo de ventanas [100].

El método MonoROCC [38] pese a estar considerado un método estéreo por la página oficial del KITTI, parece aplicable odometría monocular, usando el flujo óptico para guiarse y proponiendo un método para la detección de valores atípicos en este.

En el método S-PTAM [43], [44], para la extracción de características se usó el descriptor BRIEF [95] y el algoritmo de detección de Shi-Tomasi [96], el cual aunque se menciona que es más lento que el FAST [97], asegura una buena distribución espacial de los puntos seleccionados en la imagen.

En el método SLUP [49] optimizan la estimación de la posición gracias a puntos de referencia en el terreno, como señales de tránsito.

En el método MFI [52] y TLBBA [53] se realiza la detección de características con el algoritmo SIFT [98], pero acelerándola con una GPU (*Graphics Processing Unit* [unidad de procesamiento gráfico]) de C. Wu [99].

El método GT_VO3pt [73] puede resultar interesante en caso de que se tenga la necesidad de reconstruir estructuras subacuáticas en 3D.

2.1.3. Métodos con odometría monocular

La odometría monocular es un tipo de odometría visual, con la distinción de que utiliza una única cámara con un solo objetivo, consiguiendo con ella imágenes en 2D. Presenta desventajas frente a las cámaras estereoscópicas, especialmente por la dificultad de averiguar la escala de los objetos de la imagen, la cual se puede averiguar fácilmente en una cámara estereoscópica, sabiendo la distancia entre los dos objetivos. Una cámara molocular se tiene que valer de la altura a la que esta sobre el suelo, como mencionado en [101] y [71], la cual no da resultados tan eficientes, pues la medida puede no ser totalmente exacta según el terreno.

Un método a destacar es el DVSO (*Deep Virtual Stereo Odometry*) [31], que actualmente (a día 1 de julio de 2019) está clasificado como el mejor método de odometría monocular entre los métodos con artículos publicados referenciados por la base de datos del KITTI y como el vigesimotercer mejor método de odometría, teniendo un error medio de traslación del 0.9 % y un error medio de rotación del 0.0021 [deg·m^{-1}]. Los resultados del uso del método se pueden consultar en un vídeo de libre acceso.[4]

El método usa redes neuronales con aprendizaje semisupervisado, algo que en entornos muy diferenciados le da más ventaja que los aprendizajes totalmente supervisados. Para reducir el esfuerzo de obtener más datos para entrenar la red, la entrenan para el reconocimiento de la fotoconsistencia de imágenes estéreo. Mencionan

[4] https://www.youtube.com/watch?v=sLZOeC9z_tw&feature=youtu.be [Accedido: 30 de marzo de 2019]

que la estimación de profundidad monocular con el uso de aprendizaje profundo (*deep learning*) supervisado tiene mucho éxito, y proponen un modelo semisupervisado para ello.

Otro método monocular como el MLM-SFM [71], [72] presenta una manera para predecir la altura de la cámara con eficiencia en ambientes de conducción, a diferencia de métodos como el VISO2-S, VISO2-M y VISO2-M + GP, que la estiman por medio detección de características, cosa que se califica como poco práctica vista la irregularidad del terreno. La solución del método para hacer la estimación correcta usa modelos entrenados con datos.

Existen otros métodos como BLF, BCC y BLO [78] que también usan redes neuronales.

En el método CUDA-EgoMotion [82] se propone un algoritmo para el rastreo de píxeles, y en caso de que estos no estén visibles (por culpa de oclusiones) el método usa rastreo de características.

2.2. Comentarios generales

Como se ha podido ver en los métodos citados y también por el orden de los métodos en la Tabla 2.1, que coloca aquellos de menor error de rotación primero, en general los métodos de odometría lídar presentan una mayor eficacia que los de odometría estéreo, y estos a su vez son más eficaces que los de odometría monocular, a costa de incrementar el coste del diseño, por necesitar más componentes para recopilar más información que de lugar a las distintas odometrías.

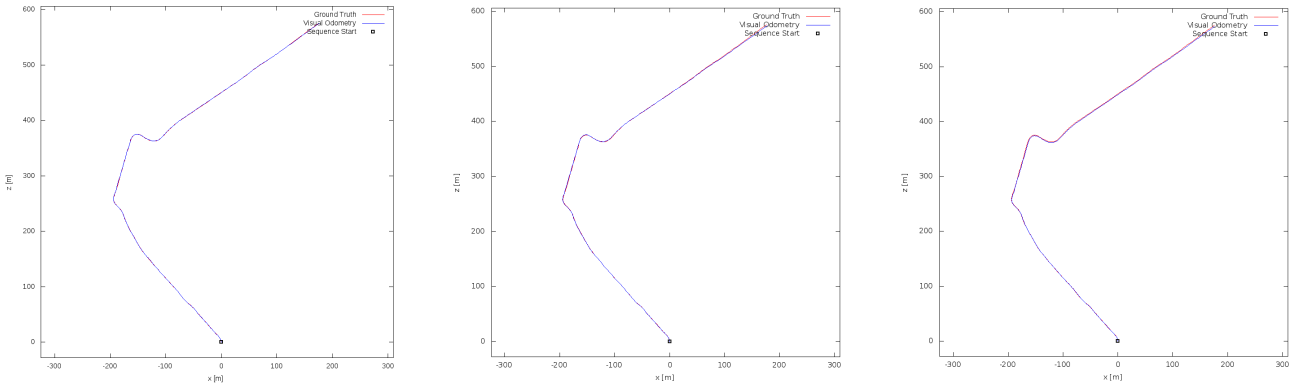
La presencia de redes neuronales en métodos de odometría monocular resultó desalentadora, pues no permitiría crear algoritmos totalmente funcionales en tiempo real y en entornos desconocidos. Afortunadamente, el método DVSO da expectativas favorables a que con un enfoque semisupervisado los algoritmos puedan funcionar en multitud de escenarios sin preparación específica. Aunque los creadores del método también mencionan que deben hacer más pruebas en distintas zonas.

Se ha podido ver también que gran cantidad de métodos se basan en la contribución entre odometría visual y lídar, y que parte de los métodos de odometría estéreo pueden usarse con odometría monocular, y viceversa, haciendo que ambas áreas puedan crecer en conjunto.

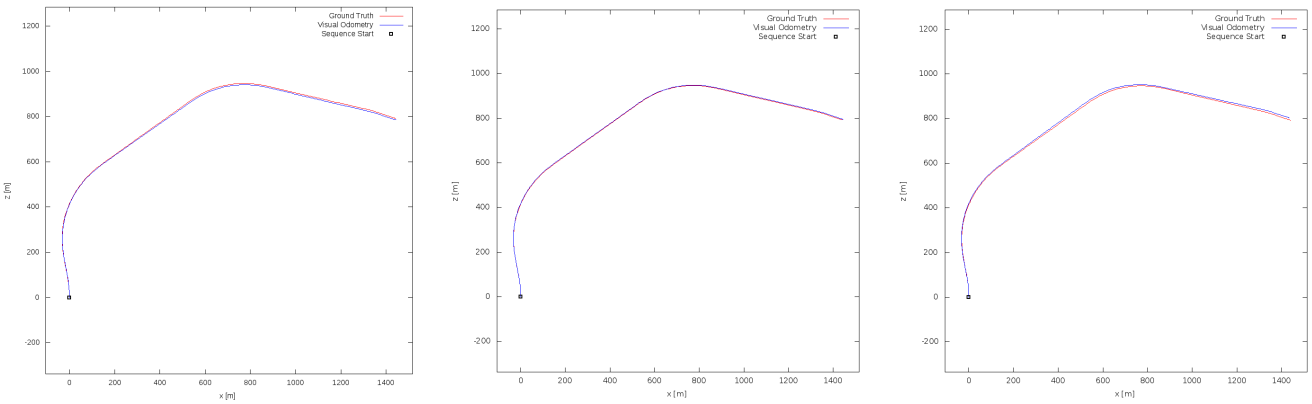
También es importante notar que la mayor, y más eficiente, en cuanto a error de traslación, parte de los métodos, están realizados en lenguaje C/C++, siendo el método BVO el de menor error de traslación que usa lenguaje Python (error del 1.76 %) y el método CUDA-EgoMotion el de menor error de rotación que usa lenguaje Matlab (error del 4.36 %).

En la Figura 2.1 se muestra el desempeño del segundo mejor método (mejor si no se tiene en cuenta al método RLO) de odometría lídar, V-LOAM, el mejor de odometría

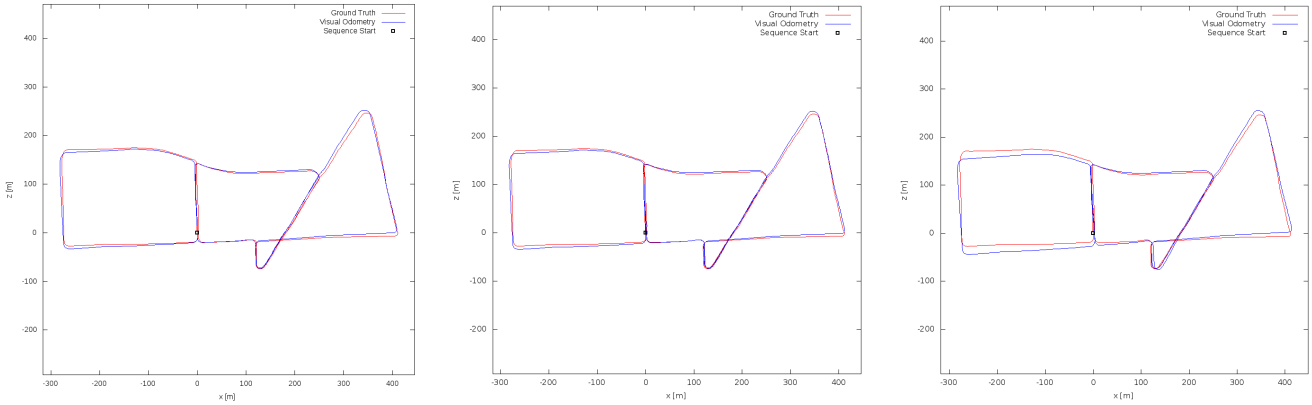
estéreo, SOFT2, y el mejor de odometría monocular, DVSO, en distintas secuencias. Es curioso notar que V-LOAM no es absolutamente mejor a los demás métodos, como puede observarse en las parte superior del trayecto de la Figura 2.1b, donde SOFT2 tiene un mejor desempeño, al predecir una trayectoria más fiel a la real que la predicha por V-LOAM. Esto es una muestra más de que hasta el actual (a día 1 de julio de 2019) mejor (o segundo mejor) método en la página web del KITTI puede ser mejorado, al menos juntando estos algoritmos presentados.



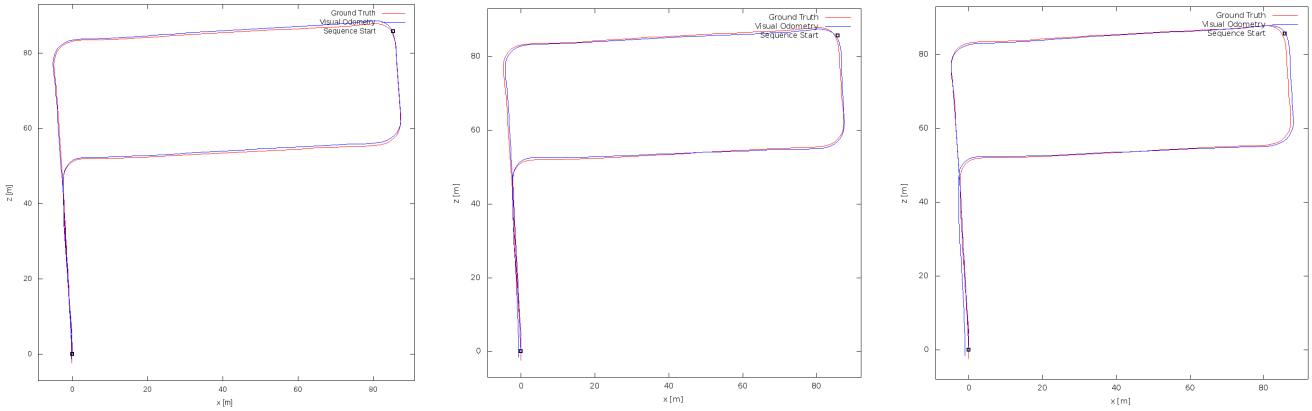
(a) Secuencia 11.



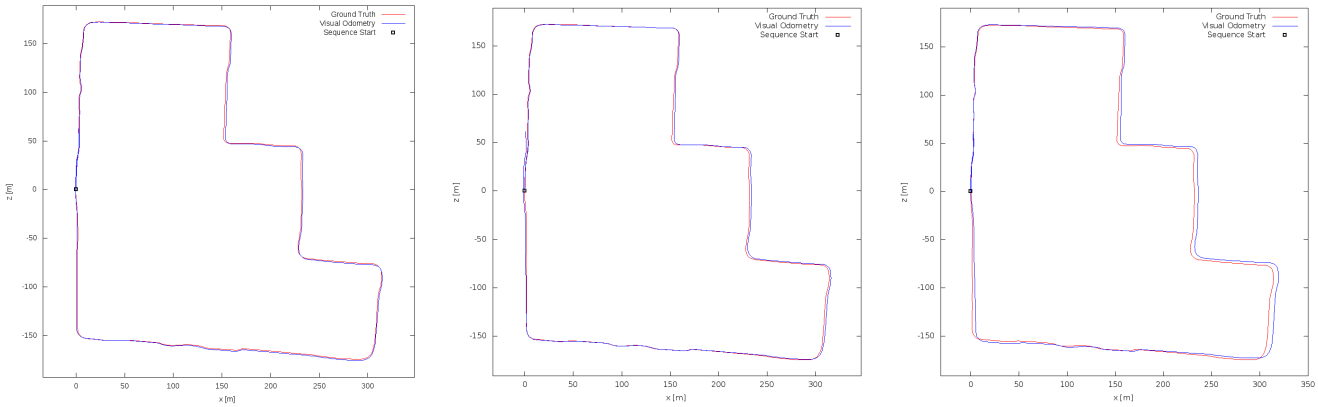
(b) Secuencia 12.



(c) Secuencia 13.



(d) Secuencia 14.



(e) Secuencia 15.

Figura 2.1: Mapas con trayectorias recorridas aproximadas por odometría visual, con los métodos V-LOAM (izquierda), SOFT2 (centro) y DVSO (derecha), en distintas secuencias de imágenes del proyecto KITTI [1]. Estas trayectorias aproximadas están coloreadas de azul, las realmente recorridas por el vehículo que registró las imágenes están coloreadas de rojo. El cuadrado negro marca el inicio del recorrido.

Capítulo 3

Programa de estimación de trayectorias

Con el objetivo final de crear un programa capaz de predecir la trayectoria que siguió una cámara que a su vez registró una secuencia de imágenes, parámetro de entrada al programa, se decidió crear un programa capaz de rastrear las trayectorias recorridas por ciertos elementos en una secuencia de imágenes. La futura realización del primer programa depende del segundo, el programa creado, cuyas características se mencionarán en este capítulo.

Para crear este programa, se usó OpenCV [5], una librería de funciones de visión por computador escrita en lenguaje C++, pero también accesible con lenguaje Python [6]. Por la facilidad de implementación que presenta Python, se prefirió hacer la implementación en este lenguaje, lo que dio lugar a un programa más lento que uno implementado en C++, sobretodo por la gran cantidad de bucles utilizados.

Varias funciones de OpenCV dependen de la librería NumPy [7], librería que facilita notablemente cálculos matriciales en Python. Por esto último, y debido a que las imágenes son matrices de puntos, también fue utilizada en el programa fuera de las funciones de OpenCV. Todo el código usado en este proyecto fue ejecutado usando la versión 3.6.7 de Python, la versión 3.4.3 de OpenCV y la versión 1.15.3 de NumPy.

Las imágenes utilizadas en las pruebas de este proyecto provinieron de dos vídeos grabados por la cámara de un teléfono móvil propio y, principalmente, de las secuencias de imágenes tomadas de las páginas del proyecto KITTI [1].

A la hora de procesar las imágenes, el color de estas no se consideró de importancia, pero sí su intensidad. Por esto, todas las imágenes procesadas fueron pasadas a escala de grises, reduciendo así cómputos matemáticos del programa.

Como se ha explicado en la Sección 1.2, se puede predecir el movimiento de una cámara entre dos imágenes grabadas por esta, conociendo los puntos correspondientes entre las imágenes (un mismo punto de un objeto presente en ambas). Para esto es importante detectar adecuadamente los puntos entre las dos imágenes, haciendo una detección y luego rastreo de las características de las imágenes. A medida que pasan distintas imágenes de la secuencia, un mismo punto se va moviendo de posición en ellas, creando una trayectoria.

Antes de presentar el programa creado como parte del proyecto, se ofrecerá unos

pequeños comentarios sobre detectores de características y buscadores de correspondencias, enfocándose en destacar los elementos que fueron utilizados por el programa realizado.

3.1. Detección de características

La detección de características es un campo en el que todavía se necesitan muchas mejoras, como se comenta en [39], especialmente para entornos de colores homogéneos y entornos de interior. Con el deseo de conseguir un rastreo eficiente de características se realizaron distintas pruebas para ver si se podían encontrar métodos de detección de características eficientes.

OpenCV tiene una sección dedicada a de detección de características [102], algunas de las cuales fueron usadas para distintas pruebas. De las pruebas realizadas se destaca el detector de Hough [103], implementado en OpenCV como una de sus variantes, [104]; Y, el detector de bordes de Canny [105], que podían presentar utilidad para la detección en escenarios muy concretos.

También se destaca al detector incluido en el descriptor ORB y al detector de Harris, en general más eficientes que los anteriormente nombrados. A continuación se aporta una descripción del detector de Harris.

3.1.1. Detector de Harris

En una imagen los puntos que habitualmente son más fáciles de identificar son aquellos que corresponden a esquinas, y por tanto son los puntos ideales para intentar localizar en distintas secuencias. Un conocido detector de esquinas es el llamado Detector de Harris, presentado por primera vez en 1988 por C. Harris y M. Stephens [88], del cual se tiene constancia de que ha sido empleado para intentar que una plataforma móvil se ubicase por medio de odometría visual, como en [2], [39], [46], [54], [75] y [82].

El detector de Harris está implementado en la librería OpenCV como la función «cornerHarris», función que es a su vez usada dentro de otra función, llamada «goodFeaturesToTrack», la cual realiza una selección de los mejores puntos captados por la primera siguiendo el criterio de selección características propuesto por J. Shi y C. Tomasi [96] (del cual también se tiene constancia de que ha sido utilizado en otros proyectos, como en [44]). En lugar de «cornerHarris» la función «goodFeaturesToTrack» puede también usar internamente la función «cornerMinEigenVal», que implementa un detector similar al de Harris. Un ejemplo de la efectividad de «goodFeaturesToTrack» puede observarse en la Figura 3.1. Se concluyó que esta función y sus dos modos de funcionamiento son eficientes para el rastreo de características.

3.2. Búsqueda de correspondencias

Para poder predecir la egomoción y pudiendo realizar una detección de características en una imagen de una secuencia, deben poderse encontrar esas mismas características en una

Figura 3.1: Esquinas detectadas mediante la función de OpenCV «goodFeaturesToTrack», con la opción «cornerHarris» coloreadas de verde y con la opción «cornerMinEigenVal» coloreadas de azul.

de las imágenes siguientes de la secuencia. Esto es hacer una búsqueda de correspondencias de características entre dos imágenes.

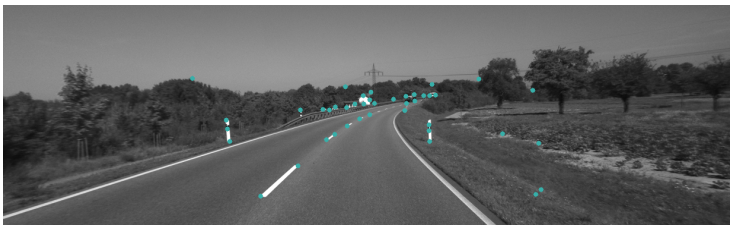
3.2.1. Flujo Óptico

Una manera de encontrar la correspondencia entre dos imágenes, es usando métodos relacionados con el flujo óptico, la percepción del movimiento entre un observador móvil y la escena que observa. También se tiene constancia de que ha sido empleado para intentar que una plataforma móvil se ubicase por odometría visual, como en [38].

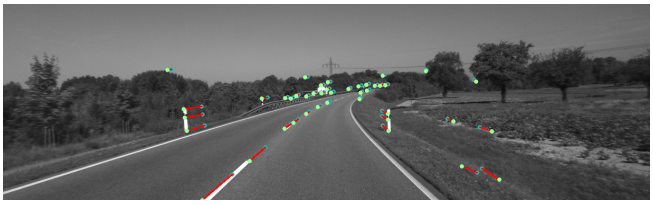
OpenCV mantiene implementadas varias funciones de detección de flujo óptico. La más interesante para este proyecto fue la función «calcOpticalFlowPyrLK», basada en el algoritmo presente en [106], una versión del rastreador de Lucas-Kanade [89]. Esta función es capaz de predecir el movimiento de unos puntos presentes en una imagen a una segunda, sin necesidad de hacer una detección de características en la segunda imagen.

Un ejemplo de la efectividad de «calcOpticalFlowPyrLK» se puede observar en la Figura 3.2, donde tras hacer una detección de puntos con «goodFeaturesToTrack» en una primera imagen de una secuencia, se halló el flujo óptico entre esta, su secuencia consecutiva y una posterior.

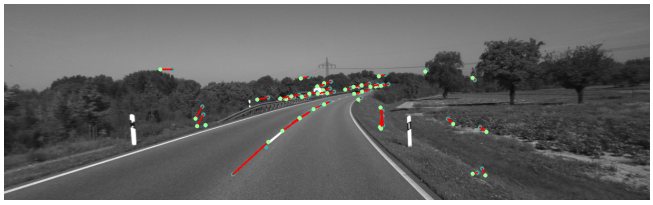
Como se puede observar, las correspondencias parecen correctas cuando no hay una gran distancia entre las imágenes de la secuencia, como en la Figura 3.2b, pero puede dar lugar a muchos errores cuando hay cierta separación, como en la Figura 3.2c en la que no se pudo hacer ni una sola detección correcta.



(a) Detección de puntos en la primera imagen de una secuencia mediante «goodFeaturesToTrack».



(b) Flujo óptico detectado entre la primera imagen de una secuencia y su consecutiva.



(c) Flujo óptico detectado (incorrectamente) entre la primera imagen de una secuencia y una posterior.

Figura 3.2: Flujo óptico entre la primera imagen de una secuencia e imágenes posteriores mediante la función de OpenCV «calcOpticalFlowPyrLK». Los puntos detectados en la primera imagen de la secuencia están coloreados de azul y están unidos con una línea roja a los correspondientes en las imágenes posteriores.

3.2.2. Descriptores

Un descriptor visual es un elemento que contiene descripciones de elementos de una imagen, por ejemplo las formas o los colores que tiene. Son especialmente útiles para la búsqueda de correspondencias, pues los elementos que rodean a un punto en una imagen son generalmente los mismos que rodearán al punto equivalente en una imagen sucesiva. Teniendo dos puntos correspondientes, sus descriptores deberían similares, o iguales.

3.2.2.1. Descriptor de intensidad de color

Existen descriptores basados en la intensidad de color de los píxeles alrededor de las correspondencias detectadas. Con estos descriptores dos puntos serían considerados idénticos si su entorno también es idéntico, o muy parecido. Por ejemplo, si un punto fue detectado alrededor de una zona de color totalmente blanco, y se teoriza que su correspondiente está en una zona totalmente negra, la correspondencia se consideraría inválida, descartándose, pues las intensidades de los colores son muy distintas entre sí.

3.2.2.2. Descriptor ORB

Otro descriptor usado para el proyecto fue el detector ORB [92], una alternativa veloz a los descriptores SIFT [98] y SURF [107], y que está implementando en OpenCV en la clase «ORB_create». La clase contiene un método que realiza detección de puntos en una imagen, por lo que a diferencia del primer descriptor mencionado, puede trabajar

independientemente. También se tiene constancia de que el descriptor ORB ha sido usado para intentar que una plataforma móvil se ubicase por medio de odometria visual, como en [20], [40] y [79].

3.2.3. Suma de diferencias absolutas (SAD)

La suma de diferencias absolutas o SAD (*Sum of Absolute Differences*) es una manera de medir que tan parecidos son dos partes de una imagen, midiendo que tan distintos son sus píxeles. Para ello, se realiza una sustracción entre los valores de cada uno los píxeles correspondientes de una parte y la otra, se calcula el valor absoluto de cada una de estas sustracciones y al final cada uno de estos valores es sumado.

Este método es útil para verificar si dos puntos distintos pueden ser considerados correspondientes, a partir de realizar la SAD entre sus descriptores. Si el resultado de esta SAD es considerado demasiado grande, implicaría que hay muchos píxeles dispares entre cada descriptor, por lo que la correspondencia quedaría descartada.

3.2.4. Estimación de la distancia entre correspondencias

Una manera de estimar la probabilidad de que dos puntos sean correspondientes es midiendo la distancia entre píxeles de estos. Si esta distancia es demasiado grande la correspondencia se podría desecharse. Considerándose una velocidad constante en todo el trayecto, la distancia máxima admitida entre correspondencias sería siempre la misma si la correspondencia fue encontrada en tomas consecutivas, si hubo una oclusión entre estas tomas la distancia máxima admitida será dos veces la original, si hubiesen dos oclusiones sería tres veces la original, y así sucesivamente hasta el máximo de oclusiones permitidas. Sin embargo, considerar una velocidad constante para los puntos es, en este caso, muy propenso a errores, por lo que para este proyecto no se eligió una distancia máxima muy restrictiva.

3.2.5. Filtro de Kalman

Para mejorar la calidad de las correspondencias detectadas, o incluso para poder establecer estas mismas correspondencias, se puede utilizar un filtro de Kalman [94] (una descripción más superficial del filtro se puede encontrar en [108]), filtro que permite realizar una estimación de lo próximo que va a hacer un sistema, si conoce los parámetros que rigen a este sistema y las salidas que este ha ido dando.

Por ejemplo, se podría modelar un sistema con una única posición descrita en un momento k por la coordenada x_k. Para predecir el valor de la coordenada en un instante futuro, $k+1$, es necesario que el filtro sepa que esta posición depende de su posición y velocidad en el instante anterior, x_k y v_k (se podría complicar este ejemplo estableciendo otras dependencias adicionales, como la aceleración, según lo requiera el sistema). La dependencia debe ser descrita a la hora de modelar el filtro mediante la matriz de transición, junto a otras matrices de ruido que describen la incertidumbre de la medida, y el filtro podrá realizar una estimación de la posición en el instante $k+1$. La predicción será todavía más exacta si el filtro va conociendo la posición en la que realmente se

estuvo en los instantes anteriores.

En este proyecto, el filtro de Kalman se intentó utilizar para mejorar la calidad de las correspondencias detectadas, utilizándose para seguir las distintas trayectorias de cada punto detectado en la imagen (cada punto detectado sigue una trayectoria hasta su desaparición de la imagen, cada posición en la trayectoria es una correspondencia). En principio, el programa creado juntaba la posición estimada por el filtro de Kalman junto con la detectada por los buscadores de correspondencias, para encontrar una correspondencia que formase parte de una trayectoria mas suavizada.

Para implementar el filtro de Kalman se utilizó la clase «KalmanFilter» de OpenCV y sus métodos, modelando el sistema con dos variables de posición, x e y, dependientes únicamente de ellas mismas y de una velocidad, considerada constante, propia de cada coordenada (v_x y v_y). También se hicieron varias pruebas usando el filtro de Kalman implementado por la librería pykalman [109].

Tras hacer pruebas con el filtro de Kalman utilizado, se prefirió desarrollar un filtro parecido a este, pero capaz de ajustar más cómodamente sus parámetros durante la ejecución del programa, este filtro es descrito en la Subsubsección 3.3.1.3.

3.3. Programa de rastreo de trayectorias

Se dispuso de un programa que, a partir de un vídeo realiza una detección de características en sus fotogramas, para luego establecer cuales de los puntos (características) detectados corresponden entre sí. A la vez de buscar estas correspondencias, el programa las almacena, para registrar todas las posiciones (puntos) en las que estuvo una misma característica en las distintas tomas de la imagen. Un pseudocódigo del programa original se puede consultar en el Algoritmo 1.

Algoritmo 1 Programa original

```
Detecta puntos en el primer fotograma
Guarda los puntos detectados en un array de puntos
for Cada fotograma del vídeo salvo el primero do
    Detecta puntos en fotograma actual
    for Cada array de puntos de fotogramas anteriores do
        Establece las correspondencias entre los puntos actuales y anteriores
        Actualiza las trayectorias con las nuevas correspondencias
        if La trayectoria va a superar las oclusiones máximas permitidas then
            Elimínala
    Guarda los puntos sin correspondencias del fotograma actual en el array de puntos
```

Este programa original realiza detecciones de puntos en los fotogramas de las secuencias por medio de la función de OpenCV «goodFeaturesToTrack», y luego les otorga un descriptor de intensidad de color. La búsqueda de correspondencias es realizaba mediante la SAD a estos descriptores. El programa también realiza un cálculo

de la velocidad entre cada correspondencia (distancia entre las correspondencias divida entre el número de tomas que las separaban) para así filtrar aquellas que estén demasiado alejadas unas de otras. Aquellos puntos que permanecen muchos fotogramas sin que se les encuentre correspondencias (superasen un parámetro de oclusiones máximo), son eliminados.

La utilidad final de este programa, no implementada, es de estimar los parámetros extrínsecos y con ello la trayectoria recorrida por la cámara. Para hacer esta estimación, el programa debería usar las correspondencias entre distintas tomas del vídeo, contenidas en las trayectorias de los puntos detectados.

Sin embargo, el programa carece de un indicador de la calidad de estas trayectorias, algo que permitiría elegir las más eficientes (aquellas que conllevasen menos errores) para calcular los parámetros extrínsecos. Un indicador de calidad también podría indicar el momento ideal en el que hacer el cálculo, aquel en el que este todas las trayectorias alcanzasen un cierto nivel de calidad. El programa también da lugar a falsas correspondencias, resultado de usar casi exclusivamente la SAD para detectarlas. Por ello, fue necesario crear un segundo programa, que también intentaría solucionar varios problemas del primero.

3.3.1. Programa de rastreo de trayectorias mejorado

El programa creado, con base el anterior (llegando a seguir la misma estructura que el Algoritmo 1), dejó de usar el detector de la función «goodFeaturesToTrack», sustituyéndolo por una función basada en el descriptor ORB, «ORBPointDescriptor», función que a su vez utiliza internamente varias funciones de OpenCV.

La búsqueda de correspondencias usando la SAD y los descriptores de intensidad de color fueron sustituidas por una función, también basada en el detector ORB, «ORBMatching». Se añadió una función, «CheckPointDistance», para eliminar las correspondencias demasiado alejadas entre sí. Su funcionamiento es similar a la presente en el programa original, pero esta toma en cuenta distintas distancias máximas según el eje coordenado del punto, y no una única distancia para ambos ejes.

También se añadió una función para suavizar las trayectorias detectadas por el programa, «Filter» (se ofrece una descripción más detallada de esta en la Subsubsección 3.3.1.3), y otra función, «CalculateQuality», para hacer un cálculo de la calidad de las trayectorias, en base a la calidad de las correspondencias y las detecciones que la conforman.

La implementación del detector y buscador de correspondencias realizada se basó en unos códigos que a su vez usaban las clases de OpenCV que implementan el descriptor ORB. Los códigos originales realizaban una detección de características, asignándoles a estas unos descriptores ORB. Luego, realizaban una búsqueda de correspondencias a partir de los descriptores detectados. Se modificaron estos códigos para que, además de devolver las detecciones, los descriptores y las correspondencias, devolviesen los parámetros de calidad de cada punto detectado y de cada correspondencia detectada. Adicionalmente, también se evitó que se otorgase a un mismo punto dos

correspondencias distintas, evento que sucedía en los códigos originales. La función de detección desarrollada fue llamada «ORBPointDescriptor» y la función de búsqueda de correspondencias fue llamada «ORBMatching». En la Subsubsección 3.3.1.1 se ofrece una descripción de la primera función, en la Subsubsección 3.3.1.2 de la segunda.

Es importante hacer notar que la documentación de OpenCV sobre el descriptor ORB es, a día de hoy (1 de julio de 2019), extremadamente reducida, haciendo difícil conocer detalles concretos de su funcionamiento sin investigar en el código fuente de su implementación.

Los resultados de la creación de este programa frente al original se pueden consultar en el Capítulo 4.

Se creó también un programa similar al presentado, que usaba el detector de flujo óptico de Lucas-Kanade en lugar del descriptor ORB. Sin embargo, este programa no resultaba tan viable pues este detector no funciona adecuadamente con fotogramas demasiado distanciados entre sí, facilitando que se rompiesen las trayectorias de las características de la imagen.

A continuación de mencionan algunas de las funciones más relevantes del programa.

3.3.1.1. Función «ORBPointDescriptor»

Esta función (más precisamente un método, al ser parte de una clase) recibe una imagen en la cual realiza una detección de características, obteniendo de ella unos puntos y sus descriptores.

La función crea una instancia de la clase de OpenCV «ORB_create». Luego, usa su método «detect», dándole como parámetro de entrada una imagen, devolviendo unos puntos clave en ella. Los puntos clave serán luego filtrados con la supresión de no-máximos o NMS (Non-Maximum Supression), filtrado que sirve para asegurarse que una misma característica solo sea detectada una vez, y que también sirve para asegurarse que se elige el punto que define de manera más destacable a la característica detectada. Para hacer la supresión el programa ordena los puntos clave en base a su calidad, directamente proporcional a su atributo «response», y define un área alrededor del punto clave de mayor calidad en función de un parámetro de distancia; Todo punto que este dentro de ese área será eliminado. Posteriormente los demás puntos que no fueron sido eliminados, también ordenados en base a su calidad, establecerán también un área que irá eliminando a otros puntos en ella. Esto acabará cuando todos los puntos hayan establecido su área o hayan sido eliminados.

Tras este filtrado, se obtendrán los descriptores de los puntos restantes, usando el método «compute», que devolverá de nuevo una matriz con los puntos clave detectados (alguno pudo ser eliminado por estar demasiado cerca del borde de la imagen) y otra matriz con sus descriptores.

Posteriormente, la función iterará sobre la matriz de los puntos clave, verificando el atributo «response» de cada punto clave, que si es menor que el umbral elegido hará

que el punto clave sea descartado. En paralelo irá almacenando los valores del atributo «response» en una matriz, que mostrará la calidad que tuvo cada punto al ser detectado.

Finalmente, el programa devolverá los puntos restantes, junto con sus descriptores y calidades.

Un pseudocódigo de este método se puede consultar en el Algoritmo 2.

Algoritmo 2 PseudoORBPointDescriptor

```
Detecta los puntos clave de una imagen
                                        ▷ Comienza a realizar la NMS
Ordena los puntos de mayor a menor calidad
for Cada punto ordenado do
    Elimina puntos cercanos a su área
                                        ▷ Se realizó la NMS
Obtén descriptores para los puntos
for Cada punto do
    if La calidad de cada punto es menor del umbral then
        Elimina el punto y su descriptor
    else
        Guarda el valor de calidad del punto
Devuelve los puntos, sus descriptores y sus calidades
```

3.3.1.2. Función «ORBMatching»

Esta función (más precisamente un método, al ser parte de una clase) recibe los puntos y descriptores detectados por la función «ORBPointDescriptor» en un primer instante (una imagen) y uno posterior a este (otra imagen), encontrando aquellos puntos que son correspondientes.

La función crea una instancia de la clase de OpenCV «BFMatcher». Luego, usa su método «match», dándole como parámetro de entrada los descriptores de los puntos en los dos distintos instantes. Este método devuelve una matriz de variables que cuentan con el atributo «distance», que determinará de que tan buena calidad es la correspondencia (mientras menor sea el atributo mejor calidad tendrá); Esas variables de la matriz también cuentan con el atributo «trainIdx», que determinará a que grupo de puntos le corresponde la variable «distance» (la matriz viene ordenada en el mismo orden en el que venían los primeros descriptores entregados, y estos a su vez en el mismo orden de los primeros puntos, por lo que conociendo la posición en la matriz de la variable «trainIdx» y la misma variable, es posible entender cuales son correspondientes).

La función itera a lo largo de la matriz de correspondencias, donde si la variable «distance» es menor que un umbral elegido, es considerada una correspondencia válida, por lo tanto, su variable «distance» es almacenada para poder ser usada para medir la calidad de la detección, junto a «trainIdx» para poder identificar la correspondencia.

Debido a que el buscador de correspondencias puede considerar correspondientes a un mismo punto (asociado al primer descriptor entregado al método «match») distintos puntos (asociados al segundo descriptor), la función verifica que la variable «trainIdx» no se repita en ninguna correspondencia. En caso de que se repita un «trainIdx», la función solo considerará la correspondencia con una mayor calidad (menor valor de «distance»).

3.3.1.3. Función «Filter»

Esta función (más precisamente un método, al ser parte de una clase) estima a donde se han movido unos puntos a partir de su posición y velocidad en un primer instante, y a partir de su posición aparente en un segundo instante, posterior al primero.

La posición de un punto en el primer instante, p_d, es la determinada por el detector de características o por el detector de correspondencias. Su posición aparente en el segundo instante, p_m es la determinada por el buscador de correspondencias. La posición del punto estimada por la velocidad, p_v es la suma de p_d al producto entre la velocidad estimada a la que se llegó a ese primer instante, v, y entre el tiempo, n, que corresponde con el número de imágenes de distancia entre el primer instante y el segundo instante considerado. Este cálculo está presente en la Ecuación 3.1.

$$p_v = p_d + v \cdot n \tag{3.1}$$

Poseyendo estas dos posibles posiciones, p_v y p_m, el filtro las suma, dando a una más peso que a la otra en función de una constante F, calculando un punto estimado p_e. Esto es reflejado en la Ecuación 3.2, y este punto estimado es lo devuelto por la función

$$p_e = p_v \cdot F + p_m \cdot (1 - F) \tag{3.2}$$

Adicionalmente, esta función permite dar distintos valores a la constante F, en función de la longevidad de la trayectoria a la que pertenece el punto del cual se quiere hacer la estimación.

Se puede observar un ejemplo del efecto del filtro, sobre puntos con nulas oclusiones entre sí, en la Figura 3.3.

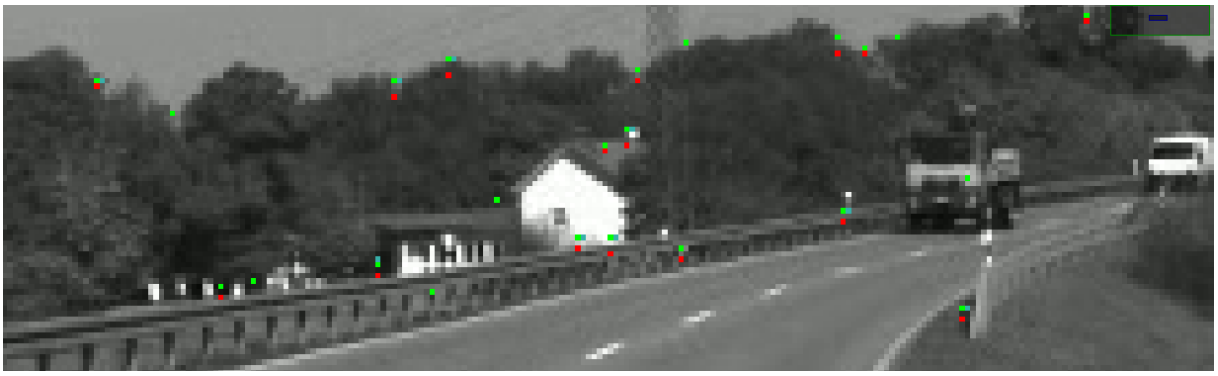


Figura 3.3: Puntos de trayectorias suavizados mediante la función «Filter» (coloreados de verde), a partir de su posición estimada por su velocidad (coloreada de rojo) y de su posición indicada por el buscador de correspondencias ORB (coloreada de azul). Algunas posiciones estimadas por el buscador ORB están tapadas por el propio punto suavizado.

Capítulo 4

Resultados del programa mejorado

Este capítulo está dedicado a mostrar los resultados de la ejecución del programa creado, descrito en el Capítulo 3, comparandolo con los resultados de la ejecución del programa original (también descrito en el Capítulo 3), el cual se usó de base para este programa modificado.

Este nuevo programa tiene de objetivo detectar las características (por ejemplo la esquina de un objeto) en una secuencia de imágenes, y encontrar sus equivalentes (el mismo objeto) en las imágenes sucesivas de la secuencia. Al encontrar los puntos equivalentes de una característica, el programa acaba creando una trayectoria, compuesta de todas las posiciones de la misma característica en distintos fotogramas. El programa original tenía este mismo principio de funcionamiento.

Una ilustración de las trayectorias detectadas por el programa creado en una secuencia de imágenes, se puede observar en la Figura 4.1, donde se asigna un color a cada trayectoria que es dibujada sobre los fotogramas.
Un parámetro importante para entender el desempeño del programa es el del número máximo de oclusiones. Este se refiere a la separación de fotogramas máxima a partir de la cual se deja considerar que un punto tendrá una correspondencia, y por tanto, haría que se diese por terminada su trayectoria. Por ejemplo, considerando un máximo de 4 oclusiones, si un punto es detectado en el fotograma 1 y no se le encuentra ninguna correspondencia entre este fotograma y el 5, el punto sería descartado.

El programa solamente considerará como trayectoria a aquella característica a la que se le haya encontrado como mínimo una correspondencia. También, el programa considerará que una trayectoria tiene longitud de 1 si tiene dos puntos correspondientes, indicando que ha pasado por dos fotogramas; Si tuviese una longitud de 2, significará que tiene tres puntos correspondientes y habría pasado por tres fotogramas; Si tuviese una longitud de 3, habría pasado por cuatro fotogramas...

El programa creado demostró efectividad para alargar las trayectorias detectadas respecto al programa original, siempre y cuando se considerase un bajo número máximo de oclusiones. Para medir cierta efectividad del programa, y considerando el número máximo de oclusiones, la manera de contar trayectorias y la manera de medir su longitud ya mencionadas; Se muestra en la Tabla 4.1 el porcentaje de mejora (de aumento de trayectorias, de su longitud media y de su varianza) del programa

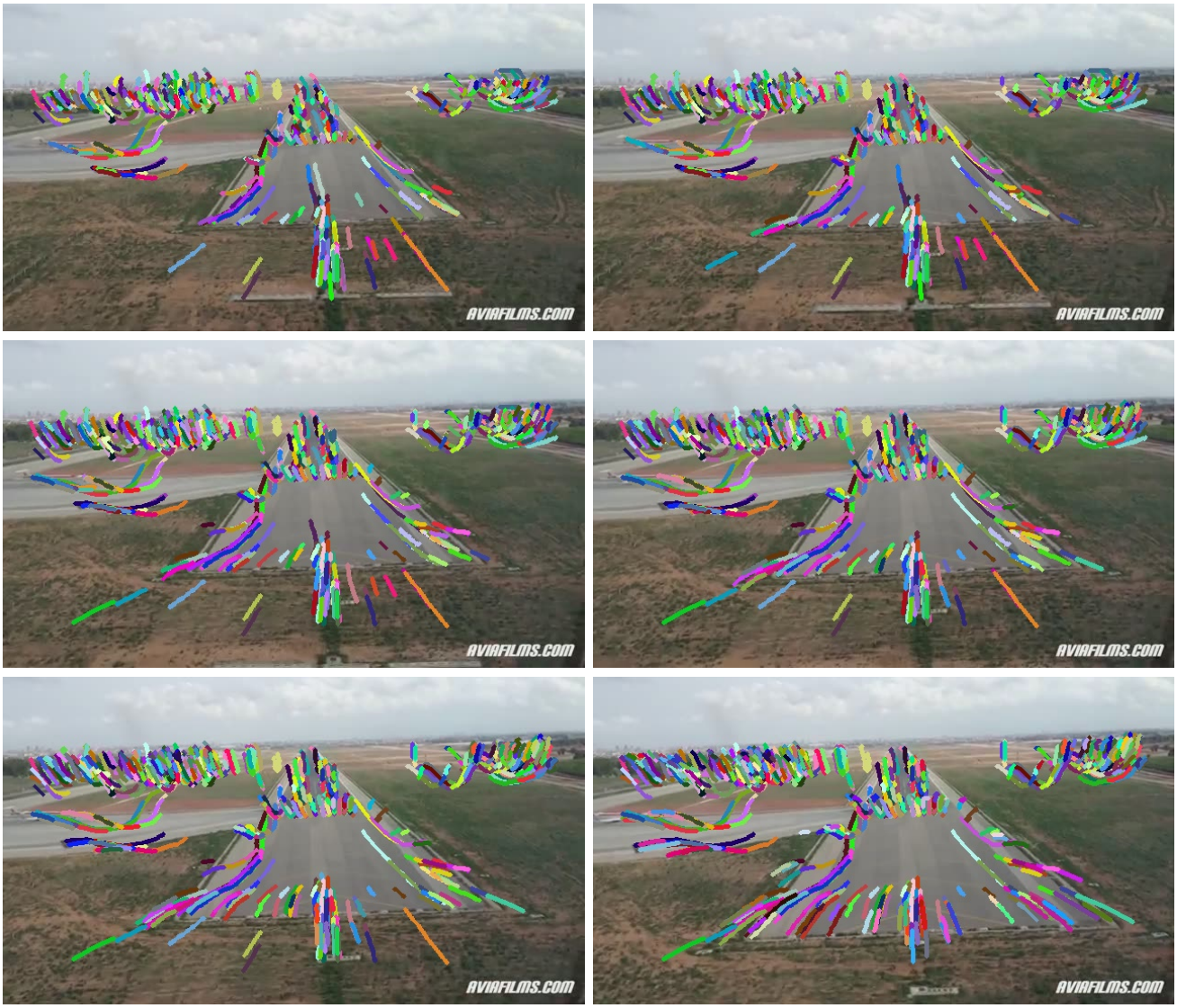


Figura 4.1: Trayectorias detectadas en una secuencia de imágenes mediante el programa realizado, en orden de aparición (de izquierda a derecha y de arriba abajo). Nótese como las trayectorias (que mantienen constante su color) intentan siempre apuntar al mismo punto en todas las imágenes.

modificado respecto al original, en varias de las secuencias del proyecto KITTI [1]. En la Tabla 4.2 se observan los datos concretos del número de trayectorias detectadas y de la media y varianza de su longitud, considerándose un máximo de 4 oclusiones; En la Tabla 4.3 considerándose un máximo de 20.

Tabla 4.1: Cambio en el número de trayectorias, junto con la media y varianza de sus longitudes, detectadas mediante el programa original y con su versión modificada en secuencias registradas por el proyecto KITTI [1]. Los porcentajes están medidos como el cambio del programa modificado frente al original.

	4 oclusiones		**20 oclusiones**			
Secuencia	**Aumento de trayectorias**	**Aumento de longitud media**	**Aumento de varianza**	**Aumento de trayectorias**	**Aumento de longitud media**	**Aumento de varianza**
4	-43.25 %	32.26 %	123.47 %	-42.71 %	-9.9 %	12.49 %
6	-53.45 %	19.44 %	131.49 %	-54.02 %	-21.47 %	48.03 %
7	-31.11 %	38.32 %	100.69 %	-36.59 %	15.42 %	142.10 %
11	-47.46 %	43.14 %	248.06 %	-49.39 %	-0.05 %	61.64 %
12	-60.07 %	37.84 %	135.38 %	-52.25 %	-10.18 %	18.45 %
13	-42.69 %	29.55 %	183.47 %	-48.31 %	-15.55 %	28.11 %
15	-48.55 %	47.45 %	244.04 %	-49.716 %	5.89 %	81.86 %
18	-34.59 %	33.413 %	147.52 %	-38.72 %	4.76 %	83.64 %
20	17.28 %	28.83 %	97.06 %	142.9 %	-18.26 %	143.99 %

Tabla 4.2: Considerando un máximo de 4 oclusiones, número de trayectorias, junto con la media y varianza de sus longitudes, detectadas con el programa original y con su versión modificada en secuencias registradas por el proyecto KITTI [1].

4 oclusiones	**Programa modificado**			**Programa original**		
Secuencia	**Trayectorias**	**Longitud media**	**Varianza**	**Trayectorias**	**Longitud media**	**Varianza**
4	14709	3.780	20.15	25917	2.858	9.017
6	44761	3.397	22.549	96166	2.844	9.741
7	59482	4.44	39.446	86349	3.21	19.655
11	43194	3.779	24.298	82213	2.64	6.982
12	40709	4.495	45.499	101951	3.261	19.933
13	164328	3.582	19.781	286731	2.765	7.804
15	92496	4.049	28.886	179787	2.746	8.396
18	95094	3.909	25.789	145381	2.930	10.419
20	45359	4.996	63.829	38677	3.878	32.391

Tabla 4.3: Considerando un máximo de 20 oclusiones, número de trayectorias, junto con la media y varianza de sus longitudes, detectadas con el programa original y con su versión modificada en secuencias registradas por el proyecto KITTI [1].

20 oclusiones	**Programa modificado**		**Programa original**			
Secuencia	**Trayectorias**	**Longitud media**	**Varianza**	**Trayectorias**	**Longitud media**	**Varianza**
4	15088	6.143	80.985	26337	6.818	71.929
6	46757	5.983	135.11	101696	7.619	91.274
7	60424	8.045	194.677	95302	6.97	80.412
11	44819	6.334	102.012	88549	6.337	61.959
12	41680	9.173	244.138	87295	10.213	206.115
13	170713	5.74	77.854	330276	6.797	60.77
15	95515	6.977	127.803	189954	6.589	70.276
18	99099	6.697	107.170	161712	6.393	58.359
20	44914	10.079	316.961	39322	8.536	129.905

Como se puede observar en las tablas, al considerar 4 oclusiones el programa realizado detecta menos trayectorias, pero a cambio aumenta, aproximadamente, entre un 20 % y 50 % sus longitudes, y posiblemente con ello sus calidades.

La razón del posible aumento de sus calidades, respecto a las del programa original, es debido a que la manera de hacer la búsqueda de correspondencias del programa original, comparando las SAD de los descriptores de los puntos detectados, era muy susceptible a errores. Por esto mismo, a pesar de que considerando 20 oclusiones el programa original sea, generalmente, mejor que el primero en cuanto a número de trayectorias detectadas, las trayectorias que el programa original detectaba no eran necesariamente muy correctas. Un número de oclusiones tan alto (respecto a la velocidad de los fotogramas por segundo de las secuencias del proyecto KITTI y la velocidad de movimiento usual del vehículo que las registró) puede dar lugar a numerosos errores, pues una trayectoria de un objeto desaparecido en la secuencia puede continuarse con uno recién aparecido, siendo esto una falsa correspondencia.

Un aumento en las detecciones y la longitud de trayectorias con tantas oclusiones no refleja necesariamente una mejora de calidad. Por esto los resultados con un número bajo de oclusiones máximas pueden indicar mejor la calidad de los programas.

El aumento de varianza del programa modificado frente al original también puede ser reflejo de que este suele dar más falsas correspondencias, haciendo que estas sean más homogéneas. Sin embargo, ambas varianzas parecen ser muy altas, producto de que algunas trayectorias se alargan demasiado en el tiempo.

Como se ha mencionado anteriormente, que una característica esté presente por más de 20 fotogramas en estas secuencias puede ser un indicio de que es una trayectoria errónea. Además de errores, situaciones específicas presentes en las secuencias pueden dar lugar a trayectorias muy largas, por ejemplo si el vehículo que grabó las secuencias se detuvo, o si este grabó un vehículo en frente suyo continuamente (porque mantenía su

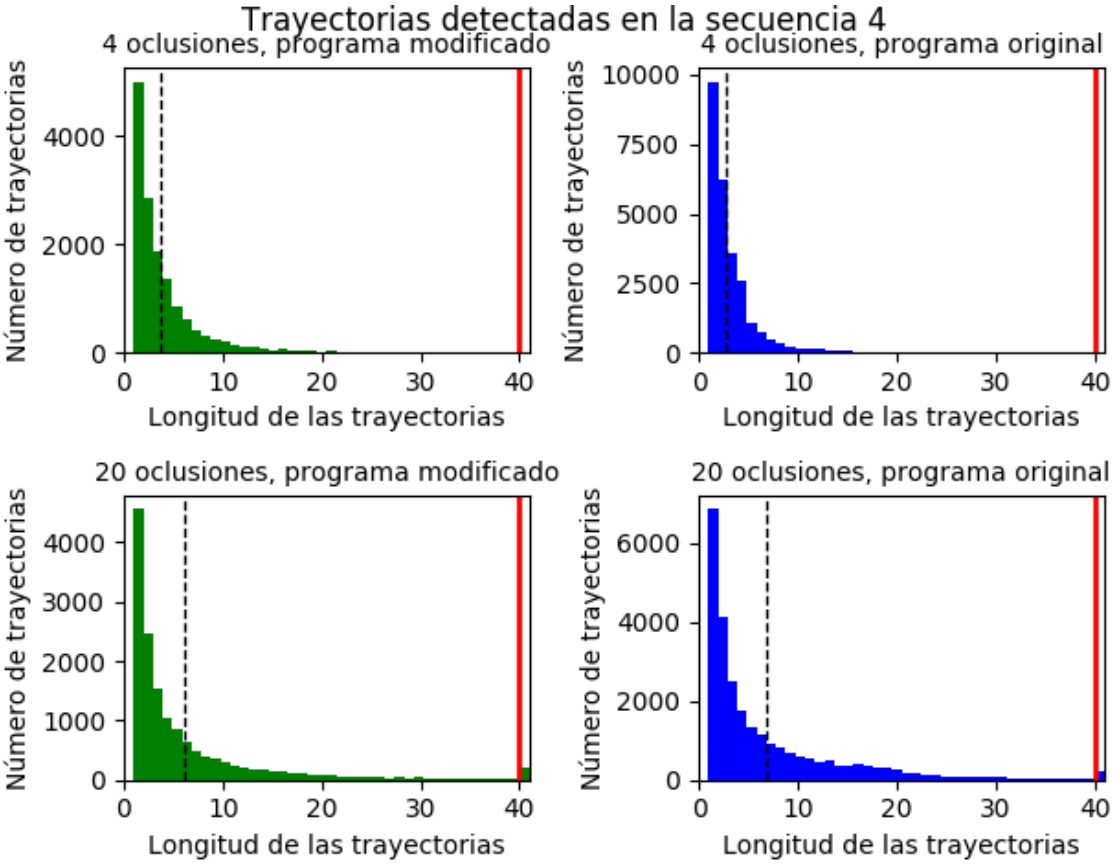
misma velocidad y dirección), o si grabó continuamente una misma montaña en el horizonte, por ejemplo. Estas situaciones justifican ligeramente la alta varianza, pero no excusan que es muy probable que haya varias trayectorias erróneas, demasiado largas, en ambos programas.

Para ver mejor la proporción de trayectorias de una cierta longitud producidas por los programas, se representan en la Figura 4.2 varios histogramas, para secuencias del proyecto KITTI, que muestran el número de trayectorias detectadas frente a su longitud. Una línea discontinua negra muestra la longitud media de las trayectorias en cada histograma. Por motivos de visualización, aquellas trayectorias mayores de 40, que son muy difíciles de ver en el histograma sin enfocar, han sido agrupadas en el límite derecho de los histogramas, como trayectorias de longitud 41, estando separadas por una línea roja.

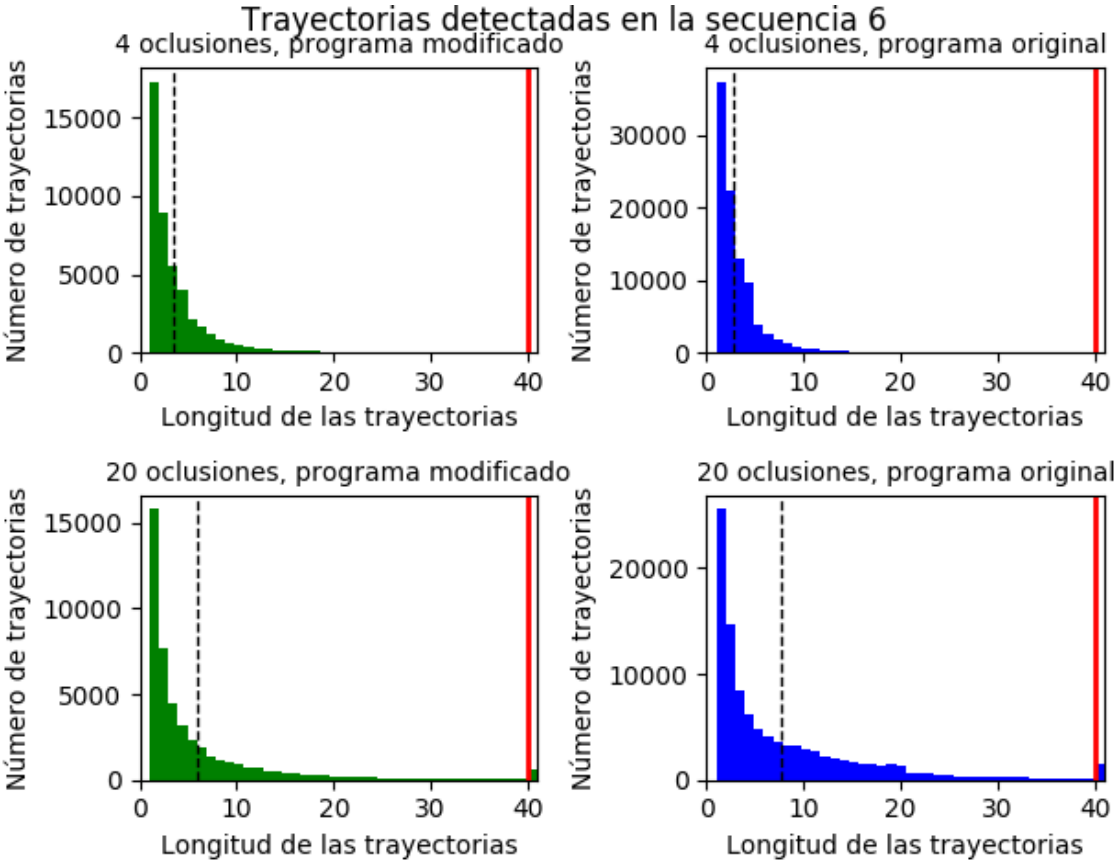
Se puede observar que en todas las secuencias los histogramas mantienen una forma similar, donde la mayor parte de las trayectorias son muy cortas, pues su número decae junto con el aumento de su longitud. También se confirma que un número alto de oclusiones puede aumentar el número de falsas correspondencias, pues en los histogramas que consideran un máximo de 20 oclusiones el número de trayectorias con longitud mayor de 40 aumenta notablemente frente a cuando solo se consideran 4 oclusiones. Se puede afirmar que una gran cantidad de estas trayectorias son erróneas, por su excesiva longitud.

Uno de los defectos principales del programa creado es que es ligeramente (depende de la secuencia y las oclusiones) más lento que el programa original, hecho que seguramente podría ser mejorado realizando el programa en lenguaje C++, por la gran cantidad de bucles que utiliza. Pese a que el lenguaje Python, y en particular la librería Numpy, reducen muy notablemente la velocidad de algunos cálculos matriciales si estos se hacen con ciertas funciones predefinidas, a veces es inevitable usar bucles, que en Python no son muy rápidos.

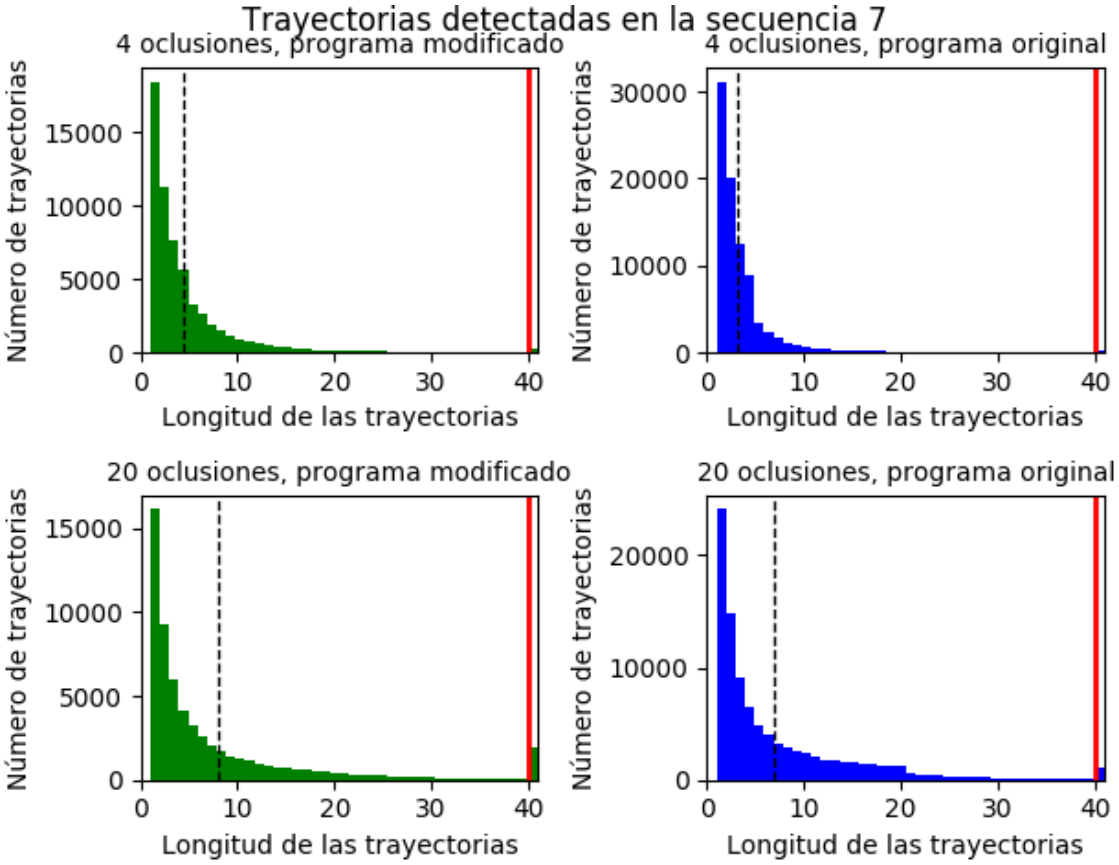
Las conclusiones finales y el trabajo futuro propuesto, continuación de este programa realizado, se pueden consultar en el Capítulo 5.


Trayectorias detectadas en la secuencia 4
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

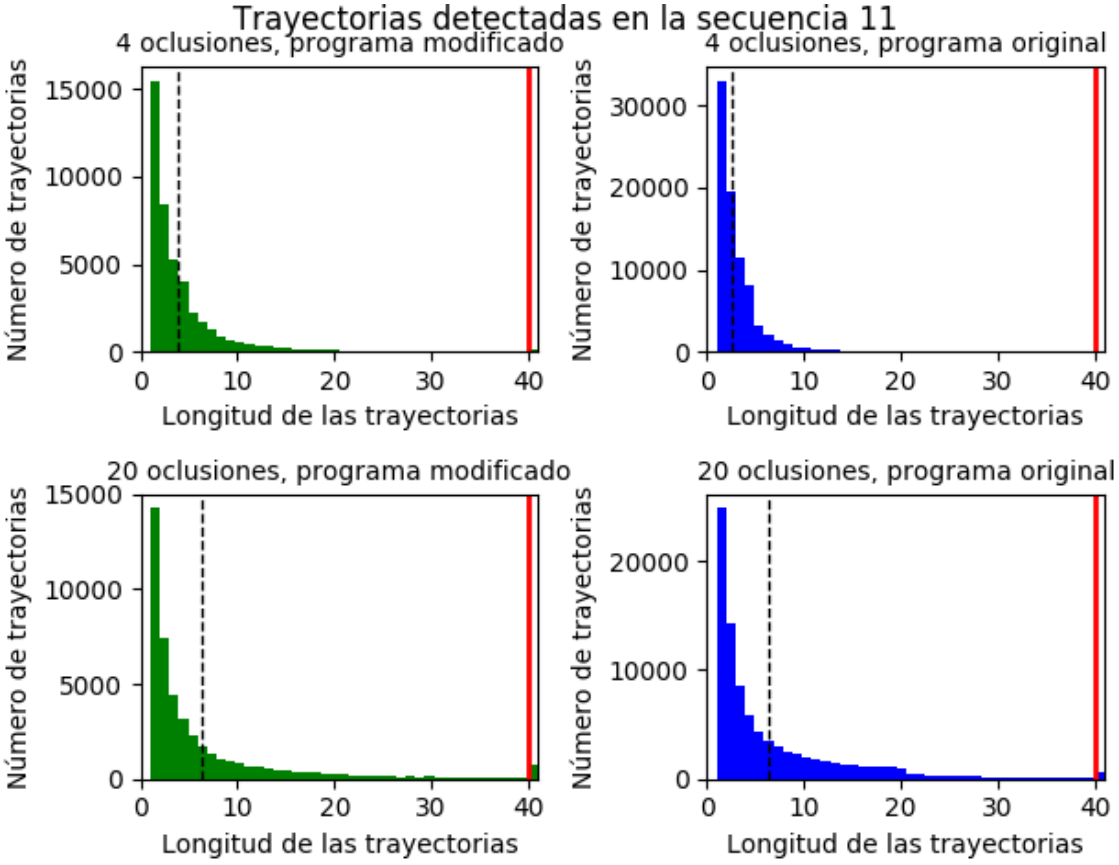
(a)


Trayectorias detectadas en la secuencia 6
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

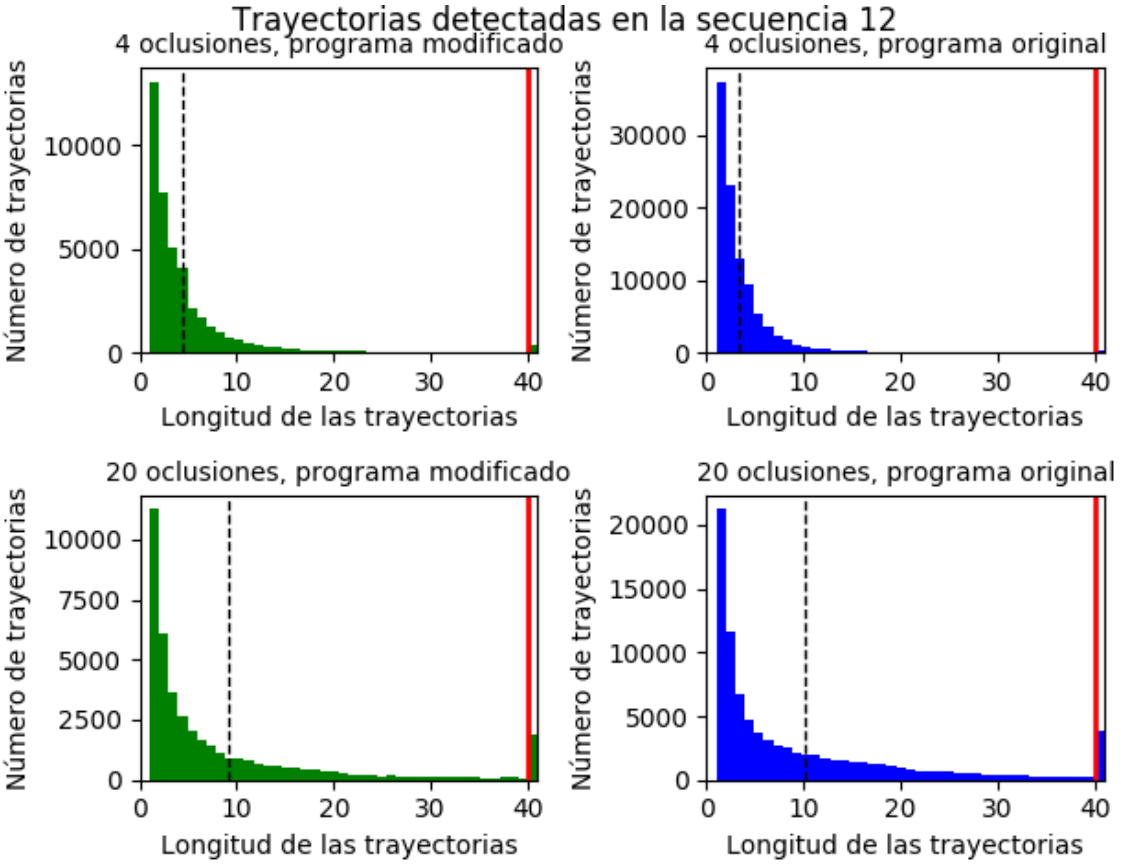
(b)


Trayectorias detectadas en la secuencia 7
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

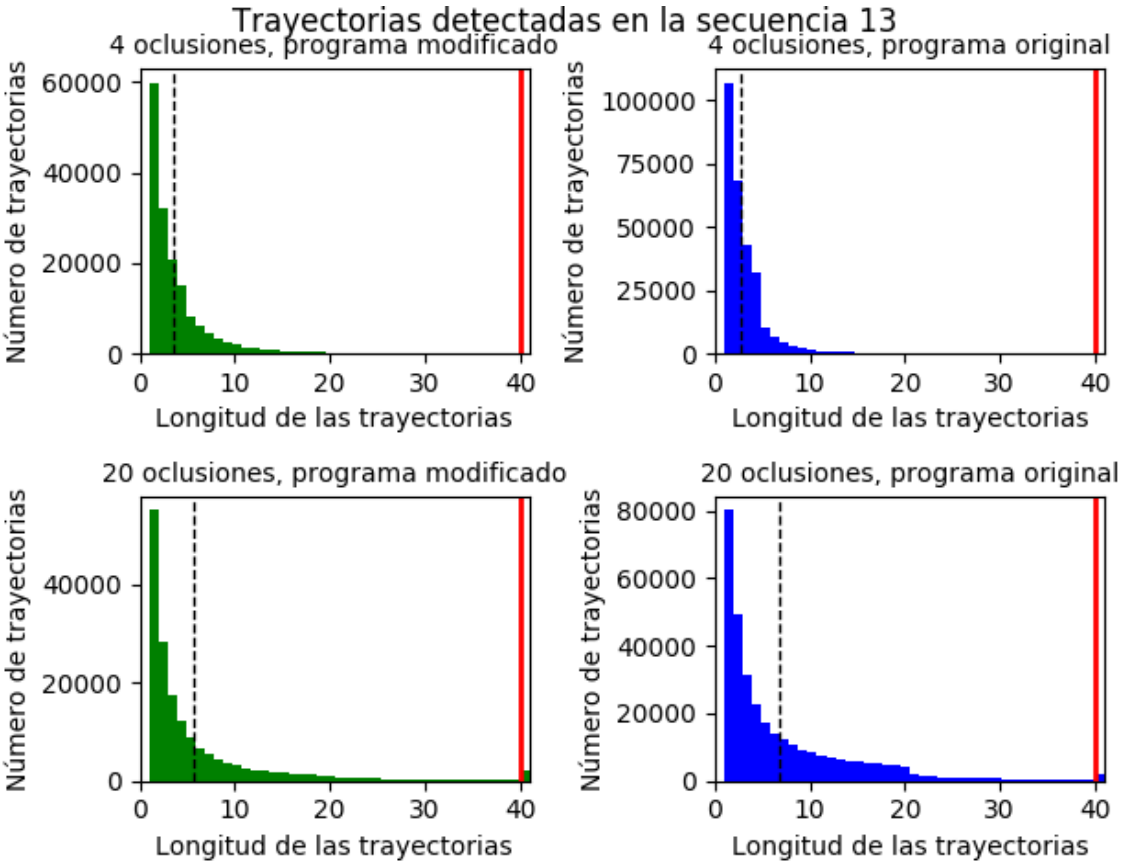
(c)


Trayectorias detectadas en la secuencia 11
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

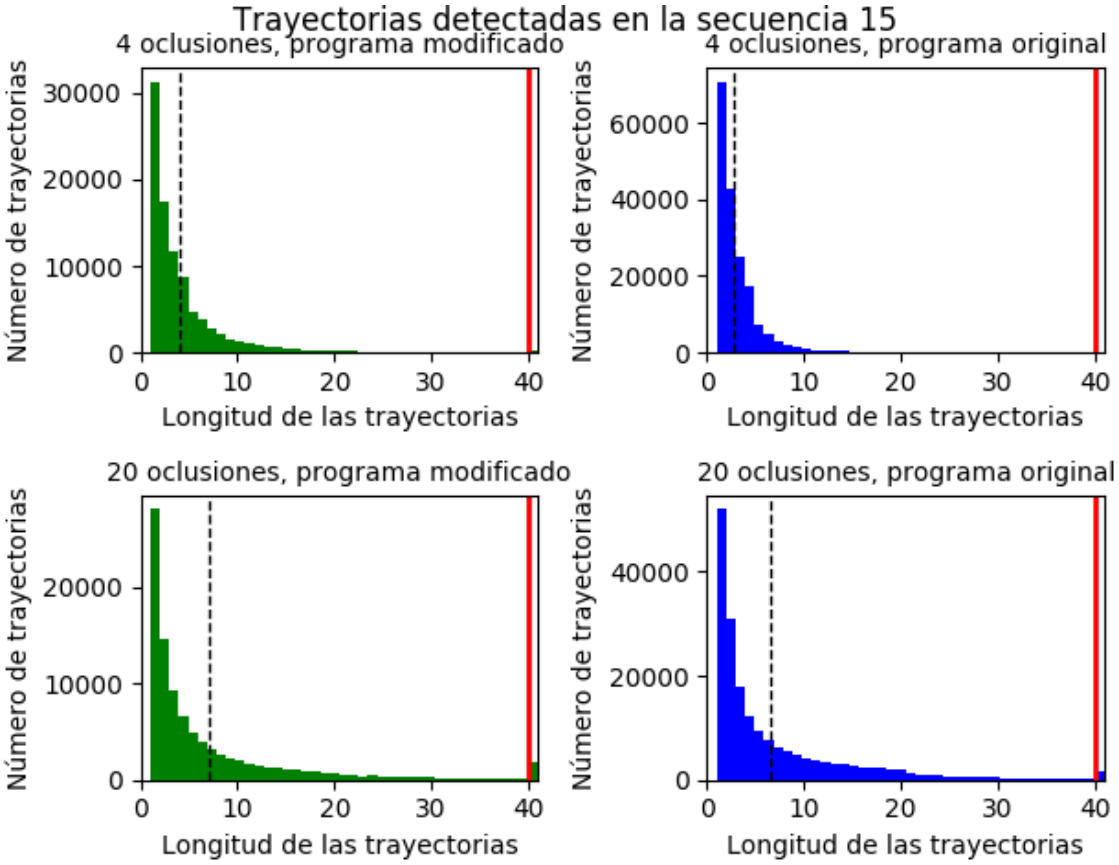
(d)


Trayectorias detectadas en la secuencia 12
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

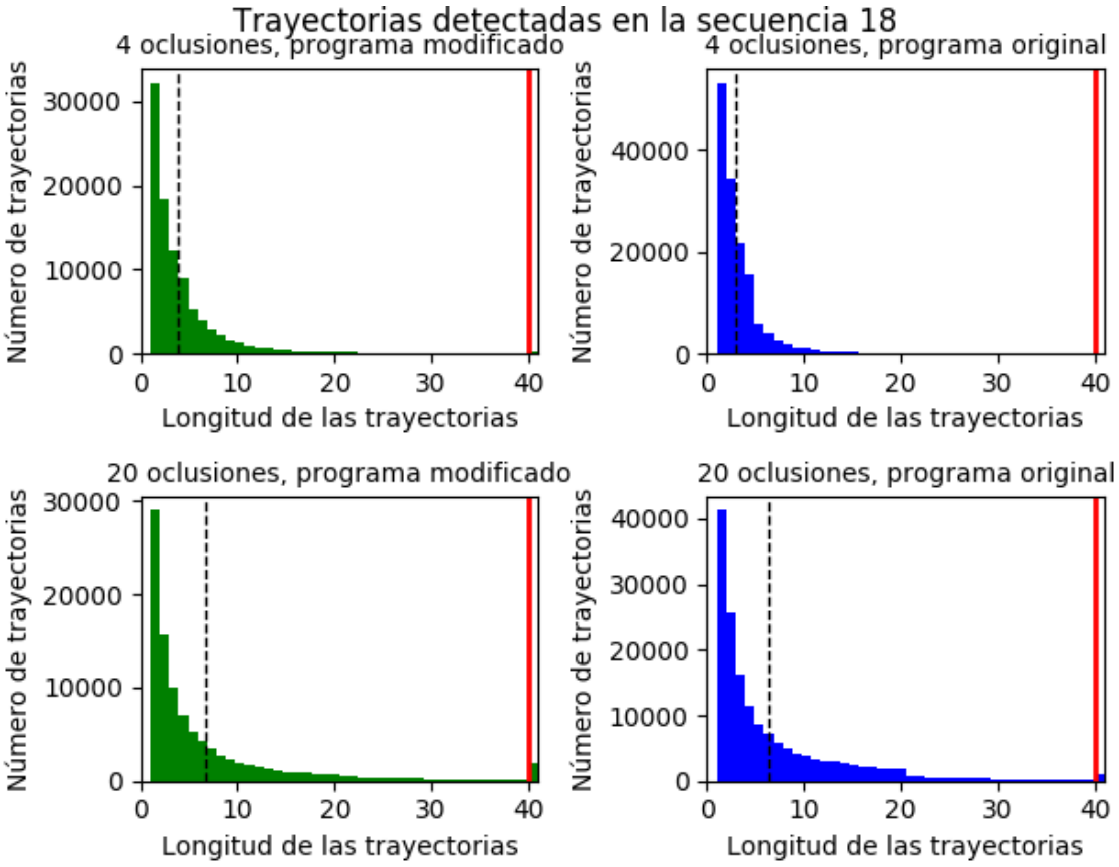
(e)


Trayectorias detectadas en la secuencia 13
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

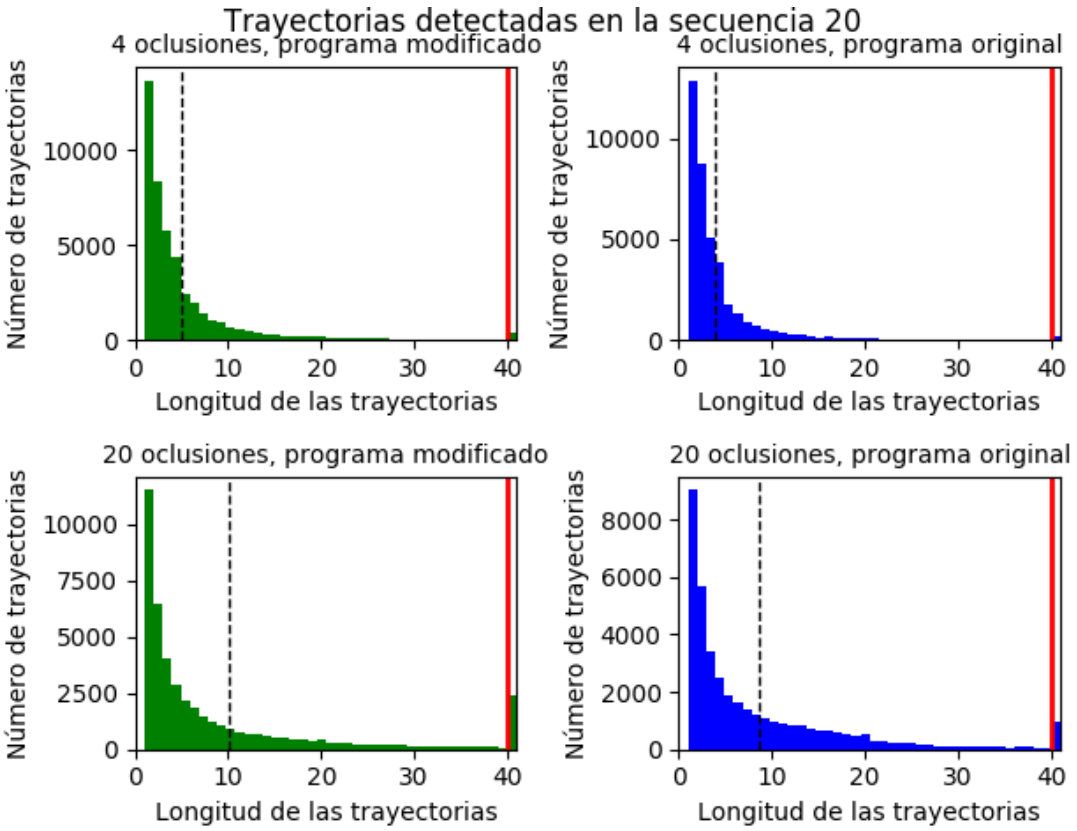
(f)


Trayectorias detectadas en la secuencia 15
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

(g)


Trayectorias detectadas en la secuencia 18
4 oclusiones, programa modificado
4 oclusiones, programa original
20 oclusiones, programa modificado
20 oclusiones, programa original
Número de trayectorias
Longitud de las trayectorias

(h)



(i)

Figura 4.2: Histogramas con el número de trayectorias detectadas en las secuencias del proyecto KITTI [1] frente a sus longitudes. La línea negra discontinua representa la longitud media de las trayectorias. Las trayectorias de longitud 41 (tras la línea roja) agrupan todas las trayectorias de longitud 41 o superior.

Capítulo 5

Conclusiones y trabajo futuro

Mediante las pruebas y el programa principal realizado, se ha podido comprobar la utilidad del detector de Harris y su efectividad, con ella entendiéndose porque es tan ampliamente usado. También se ha podido comprobar también la utilidad y efectividad del descriptor ORB, entendiendo el porqué de su presencia en métodos recientes de odometría visual.

Además de estos, se llegaron a probar una gran cantidad de detectores y descriptores, destacándose el detector de flujo óptico de Lucas Kanade. El hecho de no haber podido encontrar un digno detector ni buscador de correspondencias alternativo a los utilizados en distintos métodos de odometría visual resultó decepcionante, pero demostró que aquellos usados más usualmente son utilizados precisamente por su buena calidad.

Queda pendiente realizar un programa de estimación de la trayectoria recorrida por una cámara que utilice a su vez el programa de rastreo de trayectorias de puntos creado, y por lo tanto que realice el cálculo de los parámetros extrínsecos en función de la calidad de las trayectorias detectadas. Para ello, y posiblemente lo más complicado de implementar, es también necesario realizar una estimación de la distancia recorrida entre cada captura de la cámara, para que el mapa predicho esté correctamente escalado.

El conocer la distancia recorrida entre cada captura podría también utilizarse para mejorar las predicciones de velocidad de los puntos detectados en las imágenes, dejándola de considerar constante. Esto mejoraría el filtrado realizado por la función «Filter» programada. También, conocer está velocidad ayudaría a volver más certera la eliminación de las correspondencias con mucha distancia entre sí.

Realizar este programa de estimación de la trayectoria recorrida por la cámara permitirá ver la utilidad real del programa realizado y mencionado en este documento, dando a su vez lugar a un nuevo programa que podrá resultar de una gran utilidad para cualquier plataforma móvil.

Adicionalmente, se debe destacar que para realizar todos los programas mencionados en este documento se debieron elegir una cantidad muy grande de parámetros para cada función usada. Es muy probable que la combinación de parámetros elegidos no sea perfecta, pero la enorme cantidad de estos hace muy difícil encontrar la ideal a base de ir probando repetidamente. Por ello, desarrollar un programa de aprendizaje automatizado para encontrar los parámetros ideales sería extremadamente útil para

perfeccionar los resultados obtenidos, dando gran precisión de su ubicación a la plataforma móvil que use estos programas.

Bibliografía

[1] A. Geiger, P. Lenz, C. Stiller y R. Urtasun, «Welcome to the KITTI Vision Benchmark Suite!», en *Civlis*. [En línea]. Disponible en: http://www.cvlibs.net/datasets/kitti/index.php. [Accedido: 1 de julio de 2019]

[2] D. Nistér, O. Naroditsky y J. Bergen, «Visual Odometry», en *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. [En línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.205.9662&rep=rep1&type=pdf. [Accedido: 10 de marzo de 2019]

[3] P. F. Alcantarilla, «Vision Based Localization: From Humanoid Robots to Visually Impaired People», tesis doctoral, Universidad de Alcalá, 2011. [En línea]. Disponible en: https://core.ac.uk/download/pdf/58909784.pdf. [Accedido: 10 de marzo de 2019]

[4] A. Geiger, P. Lenz y R. Urtasun, «Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite», en *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. [En línea]. Disponible en: http://www.cvlibs.net/publications/Geiger2012CVPR.pdf. [Accedido: 10 de marzo de 2019]

[5] Open Source Computer Vision Library, «OpenCV». [En línea]. Disponible en: https://opencv.org/. [Accedido: 10 de marzo de 2019]

[6] «Python». [En línea]. Disponible en: https://www.python.org/. [Accedido: 10 de marzo de 2019]

[7] «NumPy». [En línea]. Disponible en: https://www.numpy.org/. [Accedido: 10 de marzo de 2019]

[8] D. Scaramuzza, «Tutorial on Visual Odometry», en *Robotics and Perception Group, University of Zurich*. [En línea]. Disponible en: http://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf. [Accedido: 3 de julio de 2019]

[9] «Aircraft Rotations», en *NASA Glenn Research Center*. [En línea]. Disponible en: https://www.grc.nasa.gov/www/k-12/airplane/rotations.html. [Accedido: 3 de julio de 2019]

[10] D. Oberkampf, y D. F. DeMenthon L. S. Davis, «Iterative Pose Estimation Using Coplanar Feature Points», 24 de febrero de 1995. [En línea]. Disponible en: https:

//pdfs.semanticscholar.org/36bc/490d902061a27b1c598725cd61ad9f1fd4b5.pdf. [Accedido: 2 de mayo de 2019]

[11] D. F. DeMenthon, y L. S. Davis, «Model-Based Object Pose in 25 Lines of Code». [En línea]. Disponible en: http://users.umiacs.umd.edu/~daniel/daniel_papersfordownload/Pose25Lines.pdf. [Accedido: 2 de mayo de 2019]

[12] V. Lepetit, F. Moreno-Noguer y P. Fua, «EPnP: An Accurate O(n) Solution to the PnP Problem». [En línea]. Disponible en: https://icwww.epfl.ch/~lepetit/papers/lepetit_ijcv08.pdf. [Accedido: 2 de mayo de 2019]

[13] P. Gil-Jiménez, H. Gómez-Moreno, R. J. López-Sastre y A. Bermejillo-Martín-Romo, «Estimating the focus of expansion in a video sequence using the trajectories of interest points», marzo de 2016. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0262885616300336. [Accedido: 2 de mayo de 2019]

[14] D. Nistér, «An Efficient Solution to the Five-Point Relative Pose Problem». [En línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.8769&rep=rep1&type=pdf. [Accedido: 16 de mayo de 2019]

[15] A. Saxena y S. H. Chung y A. Y. Ng, «Learning Depth from Single Monocular Images». [En línea]. Disponible en: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/saxena-nips-05.pdf. [Accedido: 16 de mayo de 2019]

[16] J. Zhang y S. Singh, «Visual-lidar Odometry and Mapping: Low-drift, Robust, and Fast», en *IEEE International Conference on Robotics and Automation (ICRA)*, 26–30 de mayo de 2015. [En línea]. Disponible en: https://frc.ri.cmu.edu/~zhangji/publications/ICRA_2015.pdf. [Accedido: 10 de marzo de 2019]

[17] R. Kümmerle et al., «On Measuring the Accuracy of SLAM Algorithms». Disponible en: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/kuemmerle09auro.pdf [Accedido: 3 de julio de 2019]

[18] M. Dimitrievski, D. Van Hamme, P. Veelaert y W. Philips, «Robust matching of occupancy maps for odometry in autonomous vehicles», en *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: VISAPP*, 2016. [En línea]. Disponible en: https://biblio.ugent.be/publication/7142759/file/7142762f. [Accedido: 14 de mayo de 2019]

[19] J. Zhang y S. Singh, «LOAM: Lidar Odometry and Mapping in Real-time», en *Robotics: Science and Systems Conference (RSS)*, 2014. [En línea]. Disponible en: https://www.ri.cmu.edu/pub_files/2014/7/Ji_LidarMapping_RSS2014_v8.pdf. [Accedido: 10 de marzo de 2019]

[20] I. Cvišić, J. Ćesić, I. Marković y I. Petrović, «SOFT-SLAM: Computationally Efficient Stereo Visual SLAM for Autonomous UAVs», en *Journal of Field Robotics*, 2017. [En línea]. Disponible en: https://pdfs.semanticscholar.org/df31/78e77fa6655bbcae65c00b732ef240d99fa5.pdf. [Accedido: 10 de marzo de 2019]

[21] J. Deschaud, «IMLS-SLAM: Scan-to-Model Matching Based on 3D Data» en *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. [En línea]. Disponible en: https://arxiv.org/pdf/1802.08633.pdf. [Accedido: 10 de marzo de 2019]

[22] F. Neuhaus, T. Koß, R. Kohnen y D. Paulus, «MC2SLAM: Real-Time Inertial Lidar Odometry using Two-Scan Motion Compensation.», en *German Conference on Pattern Recognition*, 2018. [En línea]. Disponible en: https://link.springer.com/chapter/10.1007/978-3-030-12939-2_5. [Accedido: 10 de marzo de 2019]

[23] K. Lenac, J. Ćesić, I. Marković y I. Petrović, «Exactly sparse delayed state filter on Lie groups for long-term pose graph SLAM», en *The International Journal of Robotics Research*, 2018. [En línea]. Disponible en: https://journals.sagepub.com/doi/abs/10.1177/0278364918767756. [Accedido: 10 de marzo de 2019]

[24] M. Buczko y V. Willert, «Flow-Decoupled Normalized Reprojection Error for Visual Odometry», en *19th IEEE Intelligent Transportation Systems Conference (ITSC)*, 2016. [En línea]. Disponible en: http://www.cvlibs.net/projects/autonomous_vision_survey/literature/Buczko2016ITSC.pdf. [Accedido: 10 de marzo de 2019]

[25] M. Buczko, V. Willert, J. Schwehr y J. Adamy, «Self-Validation for Automotive Visual Odometry», en *IEEE Intelligent Vehicles Symposium (IV)*, 2018. [En línea]. Disponible en: https://www.researchgate.net/profile/Martin_Buczko/publication/326426803_Self-Validation_for_Automotive_Visual_Odometry/links/5b4d183ba6fdcc8dae24622b/Self-Validation-for-Automotive-Visual-Odometry.pdf. [Accedido: 10 de marzo de 2019]

[26] M. Buczko, «Automotive Visual Odometry», 2018.

[27] J. Graeter, A. Wilczynski y M. Lauer, «LIMO: Lidar-Monocular Visual Odometry», 2018. [En línea]. Disponible en: https://arxiv.org/pdf/1807.07524.pdf. [Accedido: 10 de marzo de 2019]

[28] J. Zhu, «Image Gradient-based Joint Direct Visual Odometry for Stereo Camera», en *International Joint Conference on Artificial Intelligence, IJCAI*, 2017. [En línea]. Disponible en: https://www.ijcai.org/proceedings/2017/0636.pdf. [Accedido: 10 de marzo de 2019]

[29] K. Ji et al., «CPFG-SLAM: a robust Simultaneous Localization and Mapping based on LIDAR in off-road environment», en *IEEE Intelligent Vehicles Symposium (IV)*, 2018. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/8500599. [Accedido: 10 de marzo de 2019]

[30] I. Cvišić y I. Petrović, «Stereo odometry based on careful feature selection and tracking», en *European Conference on Mobile Robots (ECMR)*, 2015. [En línea]. Disponible en: http://www.cvlibs.net/projects/autonomous_vision_survey/literature/Cvisic2015ECMR.pdf. [Accedido: 10 de marzo de 2019]

[31] N. Yang, R. Wang, J. Stueckler y D. Cremers, «Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry», en *European Conference on Computer Vision (ECCV)*, 2018. [En línea]. Disponible en: http://openaccess.thecvf.com/content_ECCV_2018/papers/Nan_Yang_Deep_Virtual_Stereo_ECCV_2018_paper.pdf. [Accedido: 10 de marzo de 2019]

[32] J. Graeter, A. Wilczynski y M. Lauer, «Flow-Decoupled Normalized Reprojection Error for Visual Odometry», 2018. [En línea]. Disponible en: https://arxiv.org/pdf/1807.07524.pdf. [Accedido: 10 de marzo de 2019]

[33] R. Wang, M. Schwörer y D. Cremers, «Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras», en *International Conference on Computer Vision (ICCV), Venice, Italy*, 2017. [En línea]. Disponible en: http://openaccess.thecvf.com/content_ICCV_2017/papers/Wang_Stereo_DSO_Large-Scale_ICCV_2017_paper.pdf. [Accedido: 10 de marzo de 2019]

[34] D. Slepichev, M. Smirnov y E. Vendrovsky, «Realtime Stereo Visual Odometry».

[35] M. Buczko y V. Willert, «How to Distinguish Inliers from Outliers in Visual Odometry for High-speed Automotive Applications», en *IEEE Intelligent Vehicles Symposium (IV)*, 2016. [En línea]. Disponible en: https://www.researchgate.net/profile/Martin_Buczko/publication/305709493_How_to_Distinguish_Inliers_from_Outliers_in_Visual_Odometry_for_High-speed_Automotive_Applications/links/59e87971a6fdccfe7f8cf946/How-to-Distinguish-Inliers-from-Outliers-in-Visual-Odometry-for-High-speed-Automotive-Applications.pdf. [Accedido: 10 de marzo de 2019]

[36] M. Persson, T. Piccini y M. Felsberg, «Robust Stereo Visual Odometry from Monocular Techniques», en *IEEE Intelligent Vehicles Symposium*, 2015. [En línea]. Disponible en: http://www.diva-portal.org/smash/get/diva2:859674/FULLTEXT02. [Accedido: 10 de marzo de 2019]

[37] T. Qin, J. Pan, S. Cao y S. Shen, «A General Optimization-based Framework for Local Odometry Estimation with Multiple Sensors», 2019.

[38] M. Buczko y V. Willert, «Monocular Outlier Detection for Visual Odometry», en *IEEE Intelligent Vehicles Symposium (IV)*, 2017. [En línea]. Disponible en: https://www.researchgate.net/profile/Martin_Buczko/publication/315669432_Monocular_Outlier_Detection_for_Visual_Odometry/links/59e877a4458515c363112516/Monocular-Outlier-Detection-for-Visual-Odometry.pdf. [Accedido: 10 de marzo de 2019]

[39] J. Zhang, M. Kaess y M. Singh, «Real-time Depth Enhanced Monocular Odometry», en *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014. [En línea]. Disponible en: https://www.ri.cmu.edu/pub_files/2014/9/IROS_2014.pdf. [Accedido: 10 de marzo de 2019]

[40] R. Mur-Artal y J. Tardós, «ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras», en *IEEE Transactions on Robotics*, 2017. [En línea]. Disponible en: https://arxiv.org/pdf/1610.06475.pdf. [Accedido: 10 de marzo de 2019]

[41] D. Slepichev, S. Volodarskiy y E. Vendrovsky, «Realtime Stereo Visual Odometry».

[42] J. Deigmoeller y J. Eggert, «Stereo Visual Odometry without Temporal Filtering.», en *German Conference on Pattern Recognition (GCPR)*, 2016. [En línea]. Disponible en: https://link.springer.com/chapter/10.1007/978-3-319-45886-1_14. [Accedido: 10 de marzo de 2019]

[43] T. Pire et al., «S-PTAM: Stereo Parallel Tracking and Mapping», en *Robotics and Autonomous Systems (RAS)*, 2017. [En línea]. Disponible en: http://webdiis.unizar.es/~jcivera/papers/pire_etal_ras17.pdf. [Accedido: 10 de marzo de 2019]

[44] T. Pire, T. Fischer, J. Civera, P. De Cristóforis y J. Jacobo Berlles, «Stereo parallel tracking and mapping for robot localization», en *IROS*, 2015. [En línea]. Disponible en: https://zaguan.unizar.es/record/36753/files/texto_completo.pdf. [Accedido: 10 de marzo de 2019]

[45] J. Engel, J. Stückler, y D. Cremers, «Large-Scale Direct SLAM with Stereo Cameras», en *Int. Conf. on Intelligent Robot Systems (IROS)*, 2015. [En línea]. Disponible en: https://scholar.google.de/scholar?q=Large-Scale%20Direct%20SLAM%20with%20Stereo%20Cameras. [Accedido: 10 de marzo de 2019]

[46] J. Tardif, M. George, M. Laverne, A. Kelly y A. Stentz, «A New Approach to Vision-Aided Inertial Navigation», en *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 18–22 de Octubre 2010. [En línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.645.1738&rep=rep1&type=pdf. [Accedido: 10 de marzo de 2019]

[47] T. Tang, D. Yoon y T. Barfoot, «A White-Noise-On-Jerk Motion Prior for Continuous-Time Trajectory Estimation on *SE(3)*», 2018. [En línea]. Disponible en: https://arxiv.org/pdf/1809.06518.pdf. [Accedido: 10 de marzo de 2019]

[48] J. Graeter, A. Wilczynski y M. Lauer, «LIMO: Lidar-Monocular Visual Odometry», 2018. [En línea]. Disponible en: https://arxiv.org/pdf/1807.07524.pdf. [Accedido: 10 de marzo de 2019]

[49] X. Qu, B. Soheilian y N. Paparoditis, «Landmark based localization in urban environment», en *ISPRS Journal of Photogrammetry and Remote Sensing*, 2017. [En línea]. Disponible en: http://recherche.ign.fr/labos/matis/pdf/articles_revues/2017/QSP_2017_ISPRS.pdf. [Accedido: 10 de marzo de 2019]

[50] T. Tang, D. Yoon, F. Pomerleau y T. Barfoot, «Learning a Bias Correction for Lidar-only Motion Estimation», en *15th Conference on Computer and Robot Vision (CRV)*, 2018. [En línea]. Disponible en: https://arxiv.org/pdf/1801.04678.pdf. [Accedido: 10 de marzo de 2019]

[51] W. Meiqing, L. Siew-Kei y S. Thambipillai, «A Framework for Fast and Robust Visual Odometry», en *IEEE Transaction on Intelligent Transportation Systems*, 2017. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/8094876. [Accedido: 10 de marzo de 2019]

[52] H. Badino, A. Yamamoto y T. Kanade, «Visual Odometry by Multi-frame Feature Integration», en *First International Workshop on Computer Vision for Autonomous Driving at ICCV*, 2013. [En línea]. Disponible en: https://www.cv-foundation.org/openaccess/content_iccv_workshops_2013/W07/papers/Badino_Visual_Odometry_by_2013_ICCV_paper.pdf. [Accedido: 10 de marzo de 2019]

[53] W. Lu, Z. Xiang y J. Liu, «High-performance Visual Odometry with Two-stage Local Binocular BA and GPU», en *Intelligent Vehicles Symposium (IV), 2013 IEEE*, 2013. [En línea]. Disponible en: https://www.researchgate.net/profile/Zhiyu_Xiang/publication/261245806_High-performance_visual_odometry_with_two-stage_local_binocular_BA_and_GPU/links/567cb81d08ae19758384e3ce.pdf. [Accedido: 10 de marzo de 2019]

[54] I. Krešo y S. Šegvić, «Improving the Egomotion Estimation by Correcting the Calibration Bias», en *VISAPP*, 2015. [En línea]. Disponible en: http://www.zemris.fer.hr/~ssegvic/pubs/kreso15visapp.pdf. [Accedido: 10 de marzo de 2019]

[55] J. Behley y C. Stachniss, «Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments», en *Robotics: Science and Systems (RSS)*, 2018. [En línea]. Disponible en: http://roboticsproceedings.org/rss14/p16.pdf. [Accedido: 10 de marzo de 2019]

[56] D. Schlegel, M. Colosi y G. Grisetti, «ProSLAM: Graph SLAM from a Programmer's Perspective», 2017. [En línea]. Disponible en: https://arxiv.org/pdf/1709.04377.pdf. [Accedido: 10 de marzo de 2019]

[57] H. Badino y T. Kanade, «A Head-Wearable Short-Baseline Stereo System for the Simultaneous Estimation of Structure and Motion», en *IAPR Conference on Machine Vision Application*, 2011. [En línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.3938&rep=rep1&type=pdf. [Accedido: 10 de marzo de 2019]

[58] F. Bellavia, M. Fanfani, F. Pazzaglia y C. Colombo, «Robust Selective Stereo SLAM without Loop Closure and Bundle Adjustment», en *ICIAP*, 2013. [En línea]. Disponible en: https://link.springer.com/content/pdf/10.1007%252F978-3-642-41181-6_47.pdf. [Accedido: 10 de marzo de 2019]

[59] F. Bellavia, M. Fanfani y C. Colombo, «Selective Visual Odometry for Accurate AUV Localization», en *Autonomous Robots*, 2015. [En línea]. Disponible en: https://www.researchgate.net/profile/Fabio_Bellavia/publication/288323114_Selective_visual_odometry_for_accurate_AUV_localization/links/568284af08ae197583919b9b.pdf. [Accedido: 10 de marzo de 2019]

[60] M. Fanfani, F. Bellavia y C. Colombo, «Accurate keyframe selection and keypoint tracking for robust visual odometry», en *Machine Vision and Applications*, 2016. [En línea]. Disponible en: https://flore.unifi.it/retrieve/handle/2158/1071663/312747/MVA2016.pdf. [Accedido: 10 de marzo de 2019]

[61] M. Sanfourche, V. Vittori y G. Besnerais, «eVO: A realtime embedded stereo odometry for MAV applications», en *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013. [En línea]. Disponible en: https://w3.onera.fr/copernic/sites/w3.onera.fr.copernic/files/documents/conference_papers/2013_-_iros_-_evo_a_realtime_embedded_stereo_odometry_for_mav_applications.pdf. [Accedido: 10 de marzo de 2019]

[62] J. Huai, C. Toth y D. Grejner-Brzezinska, «Stereo-inertial odometry using nonlinear optimization», en *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015)*, 2015. [En línea]. Disponible en: https://www.researchgate.net/profile/Jianzhu_Huai/publication/282662762_Stereo-inertial_odometry_using_nonlinear_optimization/links/5616ea5808ae1a8880034690/Stereo-inertial-odometry-using-nonlinear-optimization.pdf. [Accedido: 10 de marzo de 2019]

[63] F. Pereira, J. Luft, G. Ilha, A. Sofiatti y A. Susin, «Backward Motion for Estimation Enhancement in Sparse Visual Odometry», en *2017 Workshop of Computer Vision (WVC)*, 2017. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/8278080/. [Accedido: 10 de marzo de 2019]

[64] A. Comport, E. Malis y P. Rives, «Accurate Quadrifocal Tracking for Robust 3D Visual Odometry», en *ICRA*, 2017. [En línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.2139&rep=rep1&type=pdf. [Accedido: 10 de marzo de 2019]

[65] M. Meilland, A. Comport y P. Rives, «Dense visual mapping of large scale environments for real-time localisation», en *ICRA*, 2011. [En línea]. Disponible en: https://www.researchgate.net/profile/Patrick_Rives/publication/221065361_Dense_visual_mapping_of_large_scale_environments_for_real-time_localisation/links/5406f7e40cf2bba34c1e7e45/Dense-visual-mapping-of-large-scale-environments-for-real-time-localisation.pdf. [Accedido: 10 de marzo de 2019]

[66] N. Fanani, A. Stuerck, M. Ochs, H. Bradler y R. Mester, «Predictive monocular odometry (PMO): What is possible without RANSAC and multiframe bundle adjustment?», en *Image and Vision Computing*, 2017. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S026288561730118X. [Accedido: 10 de marzo de 2019]

[67] H. Mirabdollah y B. Mertsching, «Fast Techniques for Monocular Visual Odometry», en *Proceeding of 37th German Conference on Pattern Recognition*

(GCPR), 2015. [En línea]. Disponible en: https://link.springer.com/chapter/10.1007/978-3-319-24947-6_24. [Accedido: 10 de marzo de 2019]

[68] N. Fanani, M. Ochs, H. Bradler y R. Mester, «Keypoint trajectory estimation using propagation based tracking», en *Intelligent Vehicles Symposium (IV)*, 2016. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/7535500/. [Accedido: 10 de marzo de 2019]

[69] N. Fanani, A. Stuerck, M. Barnada, y R. Mester, «Multimodal scale estimation for monocular visual odometry», en *Intelligent Vehicles Symposium (IV)*, 2017. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/7995955/. [Accedido: 10 de marzo de 2019]

[70] A. Geiger, J. Ziegler y C. Stiller, «StereoScan: Dense 3d Reconstruction in Real-time», 2011. [En línea]. Disponible en: http://ww.cvlibs.net/publications/Geiger2011IV.pdf. [Accedido: 10 de marzo de 2019]

[71] S. Song y M. Chandraker, «Robust Scale Estimation in Real-Time Monocular SFM for Autonomous Driving», en *CVPR*, 2014. [En línea]. Disponible en: https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Song_Robust_Scale_Estimation_2014_CVPR_paper.pdf. [Accedido: 10 de marzo de 2019]

[72] S. Song, M. Chandraker y C. Guest, «Parallel, Real-Time Monocular Visual Odometry», en *ICRA*, 2013. [En línea]. Disponible en: https://songshiyu01.github.io/pdf/icra2013_monovo.pdf. [Accedido: 10 de marzo de 2019]

[73] C. Beall, B. Lawrence, V. Ila y F. Dellaert, «3D Reconstruction of Underwater Structures», en *IROS*, 2010. [En línea]. Disponible en: https://upcommons.upc.edu/bitstream/handle/2117/12344/borrar%20Ila%20drac2.pdf%3Bjsessionid%3D9D5CB5A1F18BCC48FA7022798B297F53?sequence%3D1. [Accedido: 10 de marzo de 2019]

[74] M. Mirabdollah y B. Mertsching, «On the Second Order Statistics of Essential Matrix Elements», en *Proceeding of 36th German Conference on Pattern Recognition*, 2014. [En línea]. Disponible en: https://link.springer.com/chapter/10.1007/978-3-319-11752-2_45. [Accedido: 10 de marzo de 2019]

[75] P. F. Alcantarilla, J. Yebes, J. Almazán y L. M. Bergasa, «On Combining Visual SLAM and Dense Scene Flow to Increase the Robustness of Localization and Mapping in Dynamic Environments», en *ICRA*, 2012. [En línea]. Disponible en: https://www.researchgate.net/profile/Luis_Bergasa/publication/261503596_On_combining_visual_SLAM_and_dense_scene_flow_to_increase_the_robustness_of_localization_and_mapping_in_dynamic_environments/links/0a85e5320ca3b61609000000.pdf. [Accedido: 10 de marzo de 2019]

[76] B. Kitt, A. Geiger y H. Lategahn, «Visual Odometry based on Stereo Image Sequences with RANSAC-based Outlier Rejection Scheme», 2010. [En línea].

Disponible en: http://ww.cvlibs.net/publications/Kitt2010IV.pdf. [Accedido: 10 de marzo de 2019]

[77] R. Gomez-Ojeda, y J. Gonzalez-Jimenez, «Robust Stereo Visual Odometry through a Probabilistic Combination of Points and Line Segments», en *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016. [En línea]. Disponible en: https://riuma.uma.es/xmlui/bitstream/handle/10630/11515/icra16rgo.pdf?sequence=1&isAllowed=y. [Accedido: 10 de marzo de 2019]

[78] M. Velas, M. Spanel, M. Hradis y H. Herout, «CNN for IMU Assisted Odometry Estimation using Velodyne LiDAR», 2017. [En línea]. Disponible en: https://arxiv.org/pdf/1712.06352.pdf. [Accedido: 10 de marzo de 2019]

[79] D. Mankowitz y E. Rivlin, «Circular FREAK-ORB Visual Odometry», 17 de junio de 2015. [En línea]. Disponible en: https://arxiv.org/pdf/1506.05257.pdf. [Accedido: 10 de marzo de 2019]

[80] M. Kaess, K. Ni y F. Dellaert, «Flow Separation for Fast and Robust Stereo Odometry», en *ICRA*, 2009. [En línea]. Disponible en: https://smartech.gatech.edu/bitstream/handle/1853/38366/Kaess09icra.pdf?sequence=1&isAllowed=y. [Accedido: 10 de marzo de 2019]

[81] P. F. Alcantarilla, L. M. Bergasa y F. Dellaert, «Visual Odometry Priors for robust EKF-SLAM», en *ICRA*, 2010. [En línea]. Disponible en: https://smartech.gatech.edu/bitstream/handle/1853/38308/Alcantarilla10icra1.pdf?sequence=1&isAllowed=y. [Accedido: 10 de marzo de 2019]

[82] A. Aguilar-González, M. Arias-Estrada y F. Berry, «The Fastest Visual Ego-motion Algorithm in the West», en *Microprocessors and Microsystems*, 2019. [En línea]. Disponible en: https://www.researchgate.net/profile/Abiel_Aguilar-Gonzalez/publication/332023165_The_Fastest_Visual_Ego-motion_Algorithm_in_the_West/links/5c9e25f2299bf1116950005d/The-Fastest-Visual-Ego-motion-Algorithm-in-the-West.pdf. [Accedido: 10 de marzo de 2019]

[83] Z. Boukhers, K. Shirahama y M. Grzegorzek, «Example-based 3D Trajectory Extraction of Objects from 2D Videos», en *Circuits and Systems for Videos Technology (TCSVT), IEEE Transaction*, 2017. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/7982664. [Accedido: 10 de marzo de 2019]

[84] Z. Boukhers, K. Shirahama y M. Grzegorzek, «Less restrictive camera odometry estimation from monocular camera», en *Multimedia Tools and Applications*, 2017. [En línea]. Disponible en: https://link.springer.com/article/10.1007%2Fs11042-017-5195-7. [Accedido: 10 de marzo de 2019]

[85] D. Frost, O. Kähler y D. Murray, «Object-Aware Bundle Adjustment for Correcting Monocular Scale Drift», en *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2012. [En línea]. Disponible

en: https://ora.ox.ac.uk/objects/uuid:580d2b5d-f8a0-44bb-bb8b-cd7518b97b16/download_file?safe_filename=icra.pdf.pdf&file_format=application%2Fpdf&type_of_work=Conference. [Accedido: 10 de marzo de 2019]

[86] S. Nagashima, K. Ito, T. Aoki, H. Ishii y K. Kobayashi, «A High-Accuracy Rotation Estimation Algorithm Based on 1D Phase-Only Correlation», en *ICIAR 2007, LNCS 4633*, 2007, pp. 210–221. [En línea]. Disponible en: http://www.aoki.ecei.tohoku.ac.jp/~ito/2007-08-ICIAR.pdf. [Accedido: 14 de mayo de 2019]

[87] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard y F. Dellaert, «iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree», en *International Journal of Robotics Research*, 2012, vol. 31, pp. 217–236. [En línea]. Disponible en: http://www.cs.cmu.edu/~kaess/pub/Kaess12ijrr.pdf. [Accedido: 5 de mayo de 2019]

[88] C. Harris y M. Stephens, «A Combined Corner and Edge Detector», en *Alvey Vision Conference*, 1988, pp. 147–151. [En línea]. Disponible en: https://www.cis.rit.edu/~cnspci/references/dip/feature_extraction/harris1988.pdf. [Accedido: 20 de abril de 2019]

[89] B. D. Lucas y T. Kanade, «An Iterative Image Registration Technique with an Application to Stereo Vision», en *Proceedings of Imaging Understanding Workshop*, 1981, pp. 121–130. [En línea]. Disponible en: https://pdfs.semanticscholar.org/51fe/a461cf3724123c888cb9184474e176c12e61.pdf. [Accedido: 20 de abril de 2019]

[90] C. Tomasi y T. Kanade, «Detection and Tracking of Point Features», abril de 1991. [En línea]. Disponible en: https://cecas.clemson.edu/~stb/klt/tomasi-kanade-techreport-1991.pdf. [Accedido: 20 de abril de 2019]

[91] S. Anderson y T. D. Barfoot, «Full STEAM ahead: Exactly sparse gaussian process regression for batch continuous-time trajectory estimation on SE(3)», en *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 2015, pp. 157–164.

[92] E. Rublee, V. Rabaud, K. Konolige y G. Bradski, «ORB: an efficient alternative to SIFT or SURF», en *IEEE Int. Conf. Comput. Vision (ICCV)*, 2011. [En línea]. Disponible en: http://www.willowgarage.com/sites/default/files/orb_final.pdf. [Accedido: 20 de abril de 2019]

[93] M. Fischler y R. Bolles, «Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography», 1981. [En línea]. Disponible en: https://www.sri.com/sites/default/files/publications/ransac-publication.pdf. [Accedido: 30 de marzo de 2019]

[94] R. E. Kalman, «A New Approach to Linear Filtering and Prediction Problems», en *Transactions of the ASME–Journal of Basic Engineering*, 1960. [En línea]. Disponible en: http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf. [Accedido: 11 de mayo de 2019]

[95] M. Calonder, V. Lepetit, C. Strecha y P. Fua, «BRIEF: Binary Robust Independent Elementary Features», en *in Computer Vision ECCV*, 2010. [En línea]. Disponible en: https://www.cs.ubc.ca/~lowe/525/papers/calonder_eccv10.pdf. [Accedido: 6 de abril de 2019]

[96] J. Shi, y C. Tomasi, «Good Features to Track», en *IEEE Conference on Computer Vision and Pattern Recognition (CVPR94)*, 1994. [En línea]. Disponible en: http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf. [Accedido: 6 de abril de 2019]

[97] E. Rosten, y T. Drummond, «Machine learning for high-speed corner detection», en *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, 2006. [En línea]. Disponible en: https://www.edwardrosten.com/work/rosten_2006_machine.pdf. [Accedido: 6 de abril de 2019]

[98] D. G. Lowe, «Distinctive Image Features from Scale-Invariant Keypoints», 5 de enero de 2004. [En línea]. Disponible en: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf. [Accedido: 6 de abril de 2019]

[99] C. Wu, «SiftGPU: A GPU Implementation of David Lowe's Scale Invariant Feature Transform (SIFT)».

[100] J. Engel, J. Sturm y D. Cremers, «Semi-Dense Visual Odometry for a Monocular Camera», en *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1449–1456. [En línea]. Disponible en: https://jsturm.de/publications/data/engel2013iccv.pdf. [Accedido: 10 de marzo de 2019]

[101] S. Song, M. Chandraker y C. Guest, «High Accuracy Monocular SFM and Scale Correction for Autonomous Driving», 2016. [En línea]. Disponible en: https://songshiyu01.github.io/pdf/pami2016_monovo.pdf. [Accedido: 30 de marzo de 2019]

[102] «Feature Detection», en *Docs.OpenCV*. [En línea]. Disponible en: https://docs.opencv.org/3.0-beta/modules/imgproc/doc/feature_detection.html. [Accedido: 6 de abril de 2019]

[103] «Hough Transform» en *The University of Edinburgh, School of Informatics*. [En línea]. Disponible en: http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_review.pdf. [Accedido: 20 de abril de 2019]

[104] J. Matas, C. Calambos y J. Kittler, «Progressive Probabilistic Hough Transform». [En línea]. Disponible en: http://cmp.felk.cvut.cz/~matas/papers/matas-bmvc98.pdf. [Accedido: 20 de abril de 2019]

[105] J. Canny, «A Computational Approach to Edge Detection» en *IEEE Transactions on Pattern Analysis And Machine Intelligence*, noviembre de 1986. [En línea]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf. [Accedido: 20 de abril de 2019]

[106] J. Bouguet, «Pyramidal Implementation of the Lucas Kanade Feature Tracker». [En línea]. Disponible en: http://robots.stanford.edu/cs223b04/algo_tracking.pdf. [Accedido: 20 de abril de 2019]

[107] H. Bay, A. Ess, T. Tuytelaars y L. V. Gool, «Speeded-Up Robust Features (SURF)». [En línea]. Disponible en: http://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_00517.pdf. [Accedido: 11 de mayo de 2019]

[108] «How a Kalman filter works, in pictures», en *Bzarg*. [En línea]. Disponible en: http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/. [Accedido: 11 de mayo de 2019]

[109] «pykalman». [En línea]. Disponible en: https://pykalman.github.io/. [Accedido: 11 de mayo de 2019]

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR